“L’EXTINCTION EST LA RÈGLE, LA SURVIE EST L’EXCEPTION” : REGROUPEMENT DE MALICIELS SELON LEURS COMPORTEMENTS

par

Claire Martel

Mémoire présenté au Département d’informatique
en vue de l’obtention d’une maîtrise en génie logiciel,
grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 3 août 2018

Le 3 août 2018

le jury a accepté le mémoire de Madame Claire Martel dans sa version finale.

Membres du jury

Professeur Shengrui Wang
Directeur de recherche
Département d'informatique

Professeur Marc Frappier
Codirecteur de recherche
Département d'informatique

Frédéric Massicotte
Codirecteur de recherche
Centre canadien de réponse aux incidents cybernétiques

Professeur Richard St-Denis
Membre interne
Département d'informatique

Professeur André Mayers
Président-rapporteur
Département d'informatique

Sommaire

De nombreux maliciels sont détectés tous les jours, cependant, la plupart de ces derniers ne sont pas nouveaux mais simplement des variations de maliciels déjà connus. Du fait de toutes ces variantes, il existe de nombreuses étiquettes données par les antivirus pour une même famille de maliciels. Il faut donc trouver un moyen de classifier un maliciel inconnu avec ceux ayant le même comportement malgré la diversité des étiquettes données par les antivirus.

Le but de ce mémoire est d'effectuer le regroupement par l'analyse dynamique et l'analyse statique pour pouvoir classifier les maliciels aux comportements semblables. L'analyse dynamique consiste à exécuter le maliciel dans un environnement contrôlé et à enregistrer les traces d'exécution, tandis que l'analyse statique revient à étudier le maliciel sans l'exécuter. En utilisant et modifiant l'algorithme Malheur, cette étude vise à mettre en place un système d'agrégation de maliciels qui opère dans un temps raisonnable pour permettre le regroupement d'un grand nombre d'éléments. L'idée est donc de créer un système qui ingère des maliciels, les classifie et d'extraire un profil type pour chaque groupe afin de le réutiliser pour détecter ou se protéger des maliciels. Un objectif de cette étude est aussi de pouvoir visualiser le résultat de ce regroupement sous forme de graphes ainsi que comparer les résultats de l'agrégation avec ceux donnés par les antivirus.

Pour ce faire, nous utilisons les résultats de différentes analyses automatisées de fichiers exécutables du CCRIC comme éléments en entrée de l'algorithme de regroupement. Nous utilisons une implémentation de l'algorithme Malheur avec l'utilisation de recherche approximative pour effectuer le regroupement et nous utilisons une variation d'AVClass pour déterminer à quelle famille un maliciel appartient. Pour vérifier la précision de notre agrégation, nous utilisons le coefficient Silhouette, ainsi que

l'exactitude.

Les résultats montrent que l'utilisation de la recherche approximative allonge le temps de regroupement, mais permet de regrouper des éléments similaires mais non égaux. En effet, les groupes générés lors d'un regroupement utilisant la recherche approximative sont plus hétérogènes mais moins nombreux, indiquant un regroupement plus souple. Les maliciels dans les groupes formés se ressemblent et il est même possible de trouver des familles de maliciels reliées entre elles. De plus, les résultats indiquent que l'utilisation d'uniformisation des données a un impact négligeable sur le regroupement et ce processus est difficile à généraliser pour tous les types de données.

Mots-clés: maliciel, antivirus, AVClass, analyse statique, analyse dynamique, regroupement, Malheur, Silhouette, graphe, extraction de signature

Remerciements

Je tiens tout d'abord à remercier mes directeurs de recherche, Shengrui Wang et Marc Frappier, pour leurs explications, conseils, commentaires et nombreuses relectures durant ces deux années de maîtrise.

J'aimerais aussi remercier le Centre canadien de réponse aux incidents cybernétiques pour m'avoir offert l'opportunité d'entreprendre cette aventure. Je remercie particulièrement Frédéric Massicotte, sans qui ce mémoire n'aurait pas vu le jour, c'est grâce à sa persévérance, sa motivation, ses conseils que j'ai pu finir ce document. C'est aussi lui qui m'a montré comment tirer des leçons et apprendre de cet exercice d'écriture et de recherche. Je voudrais aussi remercier Mathieu Couture, pour ses connaissances nombreuses en tout ce qui a trait aux maliciels, grâce à qui j'ai pu résoudre plusieurs mystères qui ont parsemé mon chemin. Je tiens aussi à remercier toutes les personnes de l'équipe d'Analytique qui m'ont de nombreuses fois aidé et remonté le moral quand la motivation manquait, que ce soit avec une pause forcée d'images de chats ou avec une blague bien méritée.

Je remercie aussi Carmen Schwesinger pour le cours rapide sur Tableau qu'elle m'a donné, sans quoi je n'aurais pu réaliser les graphiques présents dans ce mémoire avec si peu de difficulté.

Enfin, j'aimerais remercier ma famille pour leur soutien durant cette aventure, que ce soit pour relire une énième fois mon mémoire ou pour m'aider à comprendre des concepts difficiles.

Abréviations

AV Antivirus

C2 Commande et contrôle (en anglais *Command & Control*)

CCRIC Centre canadien de réponse aux incidents cybernétiques

IA Indicateur d'attaque (en anglais *Indicator Of Attack*)

IO Indicateur de compromission (en anglais *Indicator Of Compromise*)

MD5 *Message Digest 5* (fonction de hachage)

PE Portable exécutable

SHA256 *Secure Hash Algorithm 256* (fonction de hachage)

Table des matières

Liste des figures

Liste des tableaux

Introduction

Contexte

Un maliciel est un logiciel qui a des intentions malveillantes. Depuis que les maliciels sont apparus, il existe des compagnies qui tentent de contrer leurs effets néfastes et de les combattre. Les analystes en cybersécurité cherchent à comprendre le comportement de maliciels en les analysant manuellement et en extrayant des actions ou parties représentatives du code ou des comportements des maliciels qui permettent de les reconnaître pour pouvoir les détecter et s'en protéger. Ces parties représentatives sont des indicateurs de compromission (IC) ou des indicateurs d'attaque (IA). Un IC est un indicateur montrant qu'un système ou réseau est infecté par un maliciel et peut être formé de bouts de code d'un maliciel, de tentatives de connexion à une adresse internet particulière que le maliciel essaye d'atteindre, de comportements particuliers comme par exemple un nom de fichier inhabituel, *etc.* Un IA est un indicateur montrant qu'un système est sur le point ou se fait attaquer par un maliciel ; l'utilisation des IA consiste donc en une approche de défense proactive. Un IA peut être formée de tentatives de connexion à un réseau par l'acteur malicieux, de connexions suspectes dans un réseau, d'un courriel d'hameçonnage, de tentative d'ingénierie sociale, *etc.* Les IC et IA permettent de reconnaître une attaque particulière, que ce soit un maliciel ou un acteur malicieux et de la différencier d'une autre démontrant un comportement différent.

Pour analyser un maliciel et en extraire des IC ou IA, les analystes en cybersécurité analysent le fichier exécutable principalement à l'aide d'analyse statique et d'analyse dynamique. L'analyse statique est une analyse basée sur le code (source, compilé, décompilé) qui consiste à étudier le code sans l'exécuter. L'analyse dynamique consiste

à exécuter le code et à en observer son comportement. Grâce à ces analyses, il est possible d'extraire des IC ou IA qui sont ensuite partagés et utilisés pour permettre de se protéger ou de détecter un maliciel.

Le but principal d'un logiciel antivirus est de détecter si un programme est un maliciel et non de déterminer à quelle famille de maliciels ce dernier appartient. Les logiciels antivirus utilisent l'empreinte d'un programme calculée grâce à une fonction de hachage (e.g., *MD5*, *SHA256*, *SHA512*) pour déterminer si ce dernier est un maliciel ou sur des signatures provenant de IC ou IA [11, 18]. Une nouvelle variante d'un maliciel existant n'est pas toujours reconnue par un antivirus si son empreinte ou son code diffère, ce qui veut dire qu'il n'est pas toujours possible de se baser sur un antivirus pour classifier un nouveau maliciel de façon précise [2, 7].

De nos jours, le nombre de maliciels apparaissant quotidiennement est tellement important qu'il est impossible de les analyser manuellement [7, 9]. Par exemple, Kaspersky Lab rapporte que 323 000 nouveaux maliciels sont détectés chaque jour par leurs logiciels en 2016, soit 13 000 de plus qu'en 2015. L'institut AV-TEST enregistre 390 000 nouveaux maliciels chaque jour [1], tandis que le CCRIC reçoit environ 300 000 maliciels ainsi que six millions de pourriels par jour. Un maliciel possède un cycle de vie qui s'apparente au cycle de vie d'un programme régulier, comme indiqué dans la figure 1.

Étape 1 : Création du programme malicieux.

Étape 2 : Mise en production, qui consiste à mettre en ligne le maliciel et faire en sorte qu'il infecte des victimes.

Étape 3 : Analyse et rétro-ingénérie par les logiciels antivirus et les compagnies en cybersécurité pour pouvoir détecter que le programme est malicieux.

Étape 4 : Mise à jour et amélioration du maliciel. En effet, il est improbable que chaque nouveau maliciel soit formé de code complètement nouveau et différent d'autres maliciels. La plupart des maliciels sont des variations d'autres maliciels créés grâce à des techniques de diversification. Par exemple, Bayer *& al.* [3] et Walenstein et Lakhotia [20] montrent que certains auteurs de maliciels

1. https://www.helpnetsecurity.com/2016/12/08/malware-detected-daily/

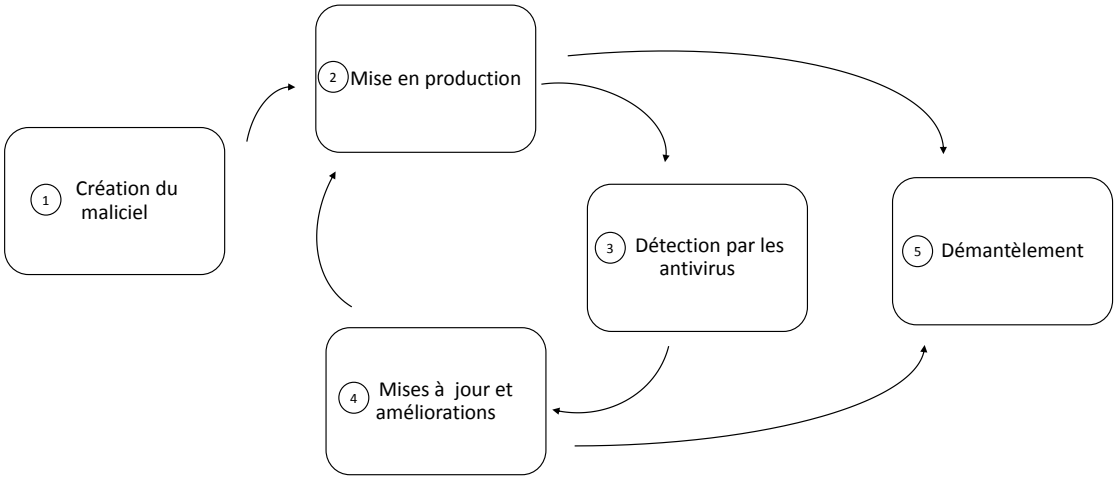


figure 1 – Cycle de vie d'un maliciel

changent la valeur d'un texte ou d'une en-tête, renomment des variables ou utilisent de l'aléatoire comme technique d'obscurcissement. De même, il est possible d'utiliser des obscurcisseurs pour empêcher l'analyse statique d'un exécutable [1, 12, 20, 22]. Ces obscurcisseurs permettent de générer de nouvelles versions d'un fichier grâce à de l'aléatoire. Aussi, Walenstein et Lakhotia [20] estiment que les développeurs de maliciels suivent des cycles de développement, ce qui implique qu'ils distribuent différentes versions de leur code au fur et à mesure qu'ils le mettent à jour et qu'ils suppriment des bogues. Les auteurs de maliciels utilisent des techniques d'obscurcissement qui changent l'aspect de leur code, ce qui leur permet de contourner les logiciels antivirus [1, 12, 20]. Après avoir effectué des modifications et améliorations sur le programme, l'auteur du maliciel procède à nouveau à une mise en production de la version mise à jour de son maliciel. Chaque mise en production est appelée une campagne.

Étape 5 : Fin du cycle de vie du maliciel en étant démantelé, soit par l'auteur du maliciel, soit par les forces de police.

Dû au nombre important de maliciels détectés chaque jour, il est important de pouvoir classifier et étiqueter les maliciels de façon automatique. En effet, les antivirus ne reconnaissent pas les nouveaux maliciels de façon automatisée, il faut donc trouver un moyen de pouvoir détecter qu'un programme est un maliciel sans se baser sur sa signature ou des bouts de son code puisque ceux-ci peuvent changer facilement. Puisque les variantes d'un même maliciel peuvent avoir des noms et signatures différents tout en ayant le même comportement ou un comportement similaire, il est important de trouver un moyen de regrouper des maliciels selon le comportement, car ceci permet d'identifier des maliciels inconnus qui sont des variantes d'une même famille [11]. Dans le même ordre d'idées, deux maliciels qui ont des signatures communes n'ont pas forcément les mêmes comportements. La technique de détection basée sur les actions du programme est donc plus résistante aux variantes puisqu'un maliciel et ses variantes se comportent de la même façon.

Pour pouvoir classifier un maliciel en fonction de son comportement, il faut l'exécuter dans un environnement contrôlé. Ainsi, il est alors possible de surveiller et enregistrer les actions entreprises par le maliciel, que ce soit au niveau du réseau, de l'hôte ou encore des bibliothèques externes que le maliciel utilise. L'étude des bibliothèques importées permet d'avoir une idée du comportement et des intentions du maliciel, car une bibliothèque est souvent importée par un auteur de maliciel dans l'intention d'être utilisée. De plus, des maliciels important les mêmes bibliothèques ont des chances d'être écrits par le même auteur ou d'avoir des comportements similaires, faisant ainsi partie d'une même famille. Bayer *& al.* [3] et Rafique et Caballero [16] montrent qu'en utilisant des techniques de regroupement basées sur le comportement, il est possible de classifier des maliciels.

Avantages et défis

L'utilisation du regroupement en fonction du comportement procure des avantages. Il permet de classifier les maliciels en fonction du comportement et ainsi détecter les maliciels de façon plus précise qu'en se basant uniquement sur sa signature ou des IC. Le regroupement permet aussi de découvrir les liens entre les différents maliciels, car deux maliciels apparentés présentent habituellement des comportements

similaires. Des familles de maliciels apparaissent et disparaissent au fil du temps et il est possible d'en observer leur évolution grâce au regroupement qui permet d'assigner à un groupe des maliciels non encore détectés en fonction de leur comportement. L'utilisation du regroupement contrairement à la classification découle de l'impossibilité de connaître toutes les actions possibles d'un maliciel. En effet, un des prérequis de la classification est de connaître à l'avance les classes dans lesquelles les éléments vont être séparés. En utilisant le comportement pour classifier des maliciels, un nombre infini d'actions et de combinaisons d'actions existe. En effet, les auteurs de maliciels créent de nouveaux maliciels tous les jours, rendant impossible un système se basant sur des classes connues à l'avance. De même, classifier des maliciels en se basant sur leur étiquette sous-entend de connaître la famille du maliciel à l'avance, ce qui est impossible dans le cas d'un maliciel non encore détecté par les antivirus. L'avantage du regroupement est donc de pouvoir assigner à un groupe un maliciel appartenant à une nouvelle famille ou démontrant un comportement nouveau de façon automatique.

De plus, regrouper les maliciels rend possible l'extraction d'un élément type. Cet archétype peut ensuite servir à extraire une signature ou des IC, qui peuvent être utiles pour reconnaître des maliciels similaires ou pour se rendre compte qu'un environnement est infecté par un maliciel. En effet, si un maliciel possède un comportement bien particulier ou par exemple s'il essaye de se connecter à une adresse particulière, il est possible de placer une alerte automatique qui préviendra un administrateur si une machine essaye de se connecter à cette adresse.

Le but du regroupement est donc de comprendre les comportements des maliciels et d'améliorer la visibilité des analystes en regroupant les maliciels en un nombre raisonnable de groupes.

En raison du nombre important de maliciels détectés chaque jour, le processus de regroupement et d'extraction de signatures doit être automatisé. De même, il faut que ce système soit assez rapide pour pouvoir effectuer l'agrégation et l'extraction de signatures en un temps raisonnable. Enfin, il doit être possible pour un humain de visualiser les résultats du regroupement, par exemple sous forme de graphes. En effet, pour qu'un analyste visualise l'évolution des groupes, il faut que ces derniers soient dans un format facilement compréhensible par un humain. De plus, visualiser les groupes permet à un analyste de rapidement déterminer si un maliciel est apparenté

à d'autres maliciels connus, facilitant ainsi son travail.

Nous explorons les questions suivantes dans ce mémoire. Est-il possible de regrouper automatiquement et efficacement les maliciels afin de caractériser les regroupements créés ? Comment regrouper les maliciels possédant de légères différences ou utilisant de l'aléatoire dans un même groupe ? Comment vérifier la précision d'un regroupement et que les groupes correspondent réellement à une famille de maliciels ? Comment extraire de façon automatisée les signatures des groupes ?

Méthodologie

Nous proposons une technique de regroupement de maliciels basée sur le comportement du maliciel. Ce type de regroupement permet de déterminer si un maliciel est apparenté à d'autres maliciels sans se baser sur sa signature ou les IC qui lui sont reliés.

Pour cerner le comportement du maliciel, nous étudions trois aspects du comportement :

- son comportement au niveau de l'hôte, c'est-à-dire ses actions locales sur la machine infectée comme par exemple la création de fichiers et la création de clés de registre ;
- son comportement au niveau du réseau, c'est-à-dire ses connections à d'autres machines à travers le réseau ;
- son comportement au niveau des en-têtes, c'est-à-dire les bibliothèques externes appelées lors de l'exécution du maliciel.

Après étude de la littérature dans le domaine du regroupement de maliciels, nous avons choisi d'utiliser Malheur [17], un algorithme permettant de regrouper de nombreux maliciels en fonction de leur comportement et ce, en un temps d'exécution de $O(k^2 \log k + n)$. Contrairement aux autres outils, Malheur a la possibilité d'effectuer un regroupement incrémental, c'est-à-dire d'ajouter des éléments à un ancien regroupement. Nous comparons l'algorithme de regroupement Malheur avec deux variations de ce dernier utilisant la recherche approximative pour comparer les éléments à regrouper.

De plus, nous organisons les résultats du regroupement en graphes visualisables

grâce au logiciel Gephi [2]. Ces graphes permettent à un opérateur humain de visualiser, naviguer et comprendre le résultat du regroupement. Pour chaque famille de maliciels, une couleur spécifique est utilisée dans les graphes, ce qui permet de repérer rapidement les familles de maliciels. La précision du regroupement est vérifiée grâce à des mesures internes et externes, soit l'indice Silhouette et l'exactitude.

Les signatures sont extraites des groupes en calculant les instructions les plus fréquentes dans un groupe. Une fois les groupes créés, la fréquence de chaque instruction pour tous les éléments du groupe est calculée et seules les instructions les plus fréquentes sont sélectionnées pour représenter le groupe et faire partie de la signature de ce dernier.

Résultats

Les résultats montrent que les trois aspects d'étude sélectionnés permettent d'obtenir un regroupement adéquat. Cependant, certains aspects se prêtent mieux au regroupement que d'autres, tel que l'aspect en-tête qui génère un regroupement de meilleure qualité avec des groupes plus homogènes. La variation des données en entrée a donc une grande influence sur le regroupement. Les données moins changeantes comme les bibliothèques externes spécifiées en en-tête se regroupent plus facilement que des données comme des adresses IP ou encore des actions sur l'hôte.

De plus, l'algorithme original produit un regroupement avec de nombreux groupes homogènes. Cependant, plusieurs groupes détectés appartiennent à la même famille de maliciels, montrant que cet algorithme n'est pas assez tolérant aux variations présentes dans les données. Au contraire, les versions de l'algorithme modifié utilisant la recherche approximative produisent moins de groupes, tout en conservant une homogénéité pertinente à l'intérieur de ces groupes.

Des graphes au format Gephi sont générés et automatiquement organisés pour que les éléments des groupes soient visuellement ensemble. De plus, chaque famille de maliciels possède une couleur donnée, constante à travers chaque regroupement, permettant ainsi de rapidement identifier les familles d'un regroupement à l'autre. Les

2. https://gephi.org/

évaluations interne et externe calculées permettent d'étudier la qualité du regroupement de façon plus formelle qu'en se basant sur l'organisation visuelle des groupes.

Enfin, les instructions les plus fréquentes sont extraites de chaque regroupement. Cependant, ces dernières ne sont pas toujours assez précises ou ne contiennent pas assez d'instructions pour les utiliser pour reconnaitre les maliciels similaires. En effet, les instructions présentes dans les éléments d'un groupe peuvent posséder des différences mineures entre elles. Procéder à une uniformisation des instructions ou à un compte reposant sur une égalité approximative serait nécessaire pour améliorer la qualité des signatures obtenues.

Structure du mémoire

Nous commençons tout d'abord par définir certains termes et notions utiles à la compréhension de ce mémoire dans le chapitre 1. Dans ce chapitre, nous expliquons donc les principaux concepts de cybersécurité nécessaires pour comprendre la problématique, ainsi que les concepts de base sur le regroupement. Le domaine de la cybersécurité est vaste et il est nécessaire de bien comprendre les caractéristiques et comportements d'un maliciel ainsi que les techniques de détection de ces derniers pour pouvoir visualiser l'importance du regroupement des maliciels. De plus, il est important de connaître les techniques de regroupement ainsi que les termes spécifiques au regroupement pour pouvoir lire le reste de ce mémoire. Nous continuons ensuite en exposant l'état de l'art sur le regroupement de maliciels en cybersécurité dans le chapitre 2, puis nous nous concentrons sur l'algorithme choisi et nous proposons des variations dans le chapitre 3. Nous présentons les données utilisées dans nos expériences, la solution proposée pour répondre aux problèmes et explorons les expérimentations et les résultats obtenus dans le chapitre 4. Enfin, la conclusion présente les limites de ce mémoire et suggère des pistes de recherche pour des travaux futurs.

Chapitre 1

Fondements

Ce mémoire traite de notions et de concepts en cybersécurité que nous définissons immédiatement. Les concepts de base en cybersécurité sont définis en premier, suivis par l'importance de l'étiquetage de maliciels dans le contexte de la cybersécurité. Par la suite, le processus de détection des maliciels est abordé avant de terminer par une définition du regroupement, des algorithmes et des mesures.

1.1 Concepts de base en sécurité informatique

1.1.1 Maliciel

Un maliciel est un logiciel ayant des intentions malveillantes, qui effectue des actions malicieuses sur un ordinateur sans le consentement de l'utilisateur [7]. Les actions peuvent être l'affichage d'un message, la destruction de certains fichiers, l'ouverture de ports ou tout autre action prédéterminée. Les maliciels possèdent plusieurs caractéristiques : vecteurs d'attaque, moyen de propagation et ont un ou plusieurs buts.

1.1.1.1 Vecteurs d'attaque

Un maliciel peut infecter un système de plusieurs façons. Certains maliciels nécessitent une action de l'utilisateur pour que le fichier malicieux soit téléchargé ou que le système soit infecté, tandis que d'autres maliciels exploitent les vulnérabilités d'un système. Les maliciels qui exploitent les vulnérabilités d'un système utilisent des vulnérabilités [1] présentes, par exemple dans une application, le système d'exploitation ou un protocole pour accéder au système cible et l'infecter. Les actions de l'utilisateur requises par certains maliciels peuvent inclure

— la sélection un lien ;
— l'accès à un courriel ;
— l'ouverture d'un fichier ;
— la visite d'une page internet qui contient du javascript.

Certains maliciels procèdent à une infection du système en plusieurs étapes. En effet, un petit programme est d'abord téléchargé grâce à l'exploitation d'une vulnérabilité ou à une action de l'utilisateur. Ce programme est petit et ne sert qu'à télécharger le maliciel, ce qui veut dire que les antivirus ne le détectent pas.

1.1.1.2 Moyens de propagation

Une fois le maliciel téléchargé, un maliciel possède plusieurs façons de se propager et de persister dans le système.

Virus – Un virus est un bout de programme qui s'intègre à un autre programme ou fichier et qui est exécuté lorsque l'utilisateur exécute le programme auquel le virus est attaché. Tout comme les virus en biologie, ils se propagent et infectent d'autres fichiers ou programmes.

Ver – Un ver est un programme qui est capable de se propager de système en système sans interaction de la part de l'utilisateur.

Cheval de Troie – Un cheval de Troie est un programme malveillant qui se cache dans un fichier ou programme qui a l'air légitime. Le programme malveillant peut être n'importe quel autre type de parasite.

1. https://nvd.nist.gov/

L'ordinateur ou le système infecté est appelé hôte.

1.1.1.3 Buts ou propriétés

Les actions des maliciels une fois avoir infecté l'hôte avec succès peuvent être :

— **Création de portes dérobées** – Les portes dérobées sont des accès non conventionnels qui permettent à la personne ayant créé la porte dérobée d'avoir accès à l'ordinateur infecté.

— **Administration à distance** (en anglais *Remote Access*) – Certains maliciels permettent de prendre le contrôle de l'ordinateur sur lequel il est installé à distance.

— **Collection d'information** – Certains maliciels ont pour but de collecter des informations comme enregistrer la frappe, espionner les programmes utilisés ou les sites visités. Par exemple, un enregistreur de frappe (en anglais *keylogger*) est un logiciel espion qui enregistre la frappe du clavier pour, entre autres, capturer des mots de passes tapés par l'utilisateur.

— **Affichage de publicités** (logiciels publicitaires, en anglais *adware*) – Certains maliciels consistent en des logiciels qui affichent des publicités sur l'écran de l'utilisateur et l'encourage à cliquer sur ces dernières.

— **Prise en otage les données personnelles de l'utilisateur** (rançongiciel, en anglais *ransomware*) – Certains maliciels cherchent à empêcher l'accès aux fichiers ou données de l'utilisateur jusqu'à ce que ce dernier paye une rançon.

— **Communication avec d'autres ordinateurs pour pouvoir orchestrer des attaques de déni de service, envoyer des pourriels,** *etc.* – La plupart du temps, les agrégats d'ordinateurs infectés obéissent à un centre de commande et de contrôle (en anglais *Command & Control ou C2*). Ce centre de contrôle envoie des instructions aux ordinateurs infectés par exemple pour orchestrer une attaque synchronisée contre une cible donnée.

— **Téléchargement d'un autre maliciel** – Certains maliciels sondent et analysent le système sur lequel ils se trouvent et téléchargent un autre maliciel compatible avec le système.

La plupart de ces actions ne sont pas à l'origine créées avec des intentions malicieuses. Par exemple, un logiciel d'administration à distance est tout d'abord un

programme légitime qui permet d'accéder à un ordinateur à distance, utilisé par exemple pour la gestion et l'administration de serveurs à distance. Leur utilisation légitime a été détournée pour servir des intérêts malicieux et permettre à une personne d'accéder à l'ordinateur de quelqu'un sans son consentement.

Un maliciel est formé d'une combinaison de buts, moyens de propagation et de vecteurs d'attaque. Par exemple, un rançongiciel procède de la façon suivante :

1. l'utilisateur reçoit un document Word, l'ouvre et autorise les macros *(vecteur d'attaque)* ;
2. une macro du document ouvre de façon invisible un terminal (ligne de commande) *(vecteur d'attaque)* ;
3. le maliciel est téléchargé par ligne de commande et exécuté *(vecteur d'attaque)* ;
4. le maliciel s'insère dans des fichiers de démarrage pour persister à un redémarrage *(moyen de propagation)* ;
5. le maliciel envoie le document Word à tous les contacts du carnet d'adresses *(moyen de propagation)* ;
6. le maliciel liste les fichiers présents et les encrypte *(but)* ;
7. un message demandant une rançon est affiché à l'utilisateur *(but)*.

Les maliciels effectuant le même type d'action ont tendance à avoir des comportements similaires. Par exemple, tous les rançongiciels ont un but commun qui est d'empêcher l'utilisateur d'accéder à leurs fichiers et de demander une rançon contre le rétablissement de cet accès.

1.1.1.4 Identification d'un maliciel : fonction de hachage

Les intentions et comportements des maliciels ne sont pas toujours visibles et évidents. Pour pouvoir identifier un maliciel, l'empreinte de celui-ci est calculée. Plusieurs fonctions de hachage existent pour calculer une empreinte, dont MD5, SHA1, SHA256, SHA512. L'empreinte calculée est souvent représentée sous forme de chaîne de caractères hexadécimaux, comme par exemple `86385447ce35c120511647c1ade5cef9`. Rappelons qu'une fonction de hachage possède les propriétés suivantes :

— fonction à sens unique, c'est-à-dire qu'il est très difficile et souvent non calculable de retrouver l'élément original à partir de l'empreinte ;

— un même élément résulte toujours en une même empreinte ;
— deux éléments différents ont une empreinte différente, cependant les fonctions de hachage possèdent des collisions [2], ce qui peut entraîner des problèmes au niveau de l'identification d'un élément ou de la sécurité de la fonction de hachage ;
— le temps de calcul de l'empreinte est négligeable et la taille de l'empreinte est fixe selon la fonction de hachage.

Puisqu'une empreinte est rapide à calculer, de taille fixe et possède peu de collisions, celles-ci sont souvent utilisées comme identification par les compagnies en cybersécurité. Un maliciel est donc souvent référé par son empreinte *MD5* ou *SHA256*.

1.1.2 Détection de maliciels

1.1.2.1 Antivirus

Les logiciels d'antivirus utilisent une empreinte numérique du maliciel ou des bouts de programme caractéristiques [1, 11, 18]. En effet, Zhang et Reeves [23] montrent qu'une séquence d'octets caractéristique d'un maliciel peut être utilisée comme signature pour reconnaître le maliciel.

1.1.2.2 Mécanisme de contournement des mesures de détection des maliciels

La plupart des nouveaux maliciels sont en fait des variantes de maliciels déjà existants, comme l'indique Walenstein *& al.* [21] en montrant que la plupart des maliciels détectés sont 99% similaires à d'autres déjà connus. Les méthodes de détection de maliciels sont contrées par le métamorphisme de maliciels [11, 23]. Par exemple, beaucoup d'auteurs de maliciels réutilisent du code déjà existant pour empêcher la reconnaissance de leur maliciel par des antivirus [1, 7, 12, 20, 23]. Il existe différentes techniques qui leur permettent de changer leur programme de façon à ce que ce dernier ne soit plus reconnu par les antivirus. Le développement de maliciels suit un cycle

2. http://www.mscs.dal.ca/~selinger/md5collision/, https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html, https://shattered.io/

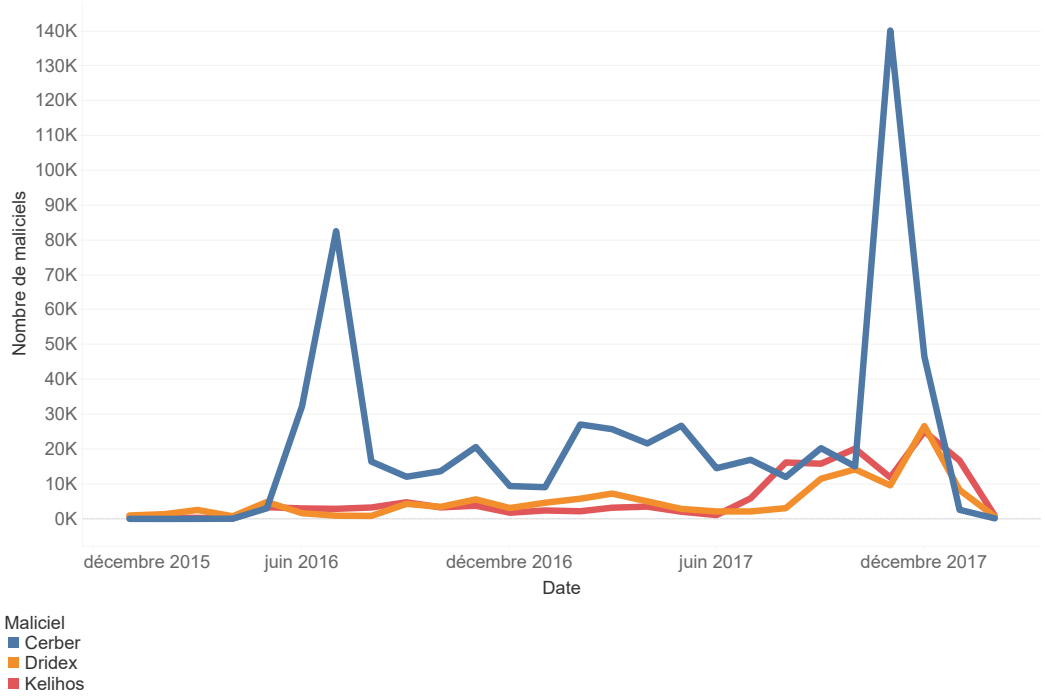


figure 1.1 – Exemple de campagnes de maliciels

de développement comme tous les autres logiciels, ce qui veut dire que les auteurs de maliciels rendent public une version de leur maliciel, puis effectuent des modifications, corrigent des bugs ou mettent à jour certaines bibliothèques avant de rendre public une nouvelle version [20]. Ces deux versions d'un même maliciel peuvent comporter de grandes différences et ne sont pas toujours reconnues comme le même maliciel par les antivirus. De même, il est possible de modifier des noms de variables, des *strings* ou même les en-têtes pour changer le programme compilé final, qui est alors différent et reconnu comme une entité nouvelle par les antivirus [20].

On appelle campagne les différentes versions itératives publiées et relâchées. Chaque campagne est représentée par une augmentation de l'activité et de la détection du maliciel, puis une diminution de l'activité du maliciel marque la fin de la campagne. La figure 1.1 montre les campagnes de trois différents maliciels. Sur cette figure, il est possible de remarquer que le maliciel *Cerber* a eu des campagnes de juin à juillet 2016, puis de novembre à décembre 2017.

D'autre part, certains auteurs de maliciels ajoutent des instructions inutiles dans

leur programme pour en cacher le but véritable et pour déjouer les antivirus. En effet, Payer [12] et Walenstein et Lakhotia [20] indiquent qu'il est possible d'ajouter des instructions inutiles ou qui ne changent pas le comportement du programme pour le rendre plus difficile à lire. En utilisant de l'aléatoire dans ces instructions ainsi créées, il est possible de générer des exécutables distincts à chaque compilation, rendant le maliciel difficile à détecter par un logiciel antivirus. Il existe même des outils qui diversifient le maliciel, par exemple en y ajoutant automatiquement des instructions ou encore en changeant des octets et génèrent un fichier exécutable différent à chaque fois [12].

Obscurcisseurs (en anglais *Packers*) L'utilisation d'obscurcisseurs permet aussi d'obscurcir le code et de le rendre illisible sans le décrypter avec l'obscurcisseur associé, ce qui rend l'analyse manuelle difficile [1, 7, 11, 12, 20, 22]. Un obscurcisseur est un logiciel qui compresse ou encrypte un autre programme, le rendant ainsi illisible sans un logiciel permettant d'inverser l'obscurcissement pour revenir au code original. Lors de l'exécution, l'obscurcissement du programme est inversé en mémoire, ce qui permet alors d'exécuter le code original. Cette technique permet d'éviter la détection du maliciel par les antivirus puisque le code du maliciel est caché. Il existe de nombreux obscurcisseurs différents et selon Yan *& al.* [22], 92% des maliciels utilisent des obscurcisseurs. Il existe plusieurs types obscurcisseurs :

1. les compresseurs qui ne font que compresser le programme original sans utiliser des techniques anti-inversion de l'obscurcissement ;
2. les encrypteurs qui encryptent et obscurcissent le code original sans le compresser ;
3. les protecteurs qui combinent encryption et compression ;
4. les groupeurs (en anglais *bundlers*) qui combinent plusieurs fichiers ou logiciels en un seul fichier exécutable.

De plus, les obscurcisseurs encryptent le maliciel original, rendant la reconnaissance de séquences d'octets par les antivirus impossible sans inverser l'obscurcissement [20]. Les obscurcisseurs sont depuis longtemps utilisés pour réduire la taille de

programmes ou pour empêcher le piratage de programmes [3].

Familles de maliciels Le métamorphisme de maliciel et les techniques d'obscurcissement créent donc de nombreux maliciels dont les actions sont les mêmes [11]. On dit alors que ces maliciels appartiennent à la même famille de maliciels. Chaque famille possède un nom associé, ce qui permet de regrouper les maliciels ayant le même code de base. Une famille de maliciels possède des caractéristiques qui permettent de la reconnaître, nommées indicateurs de compromission (IC). Les techniques d'obscurcissement rendent la détection de familles de maliciels par les antivirus difficile. Par exemple, le rançongiciel nommé «*cerber*» tente de récupérer les pages HTTP freegeoip.net, ip-api.com et ipinfo.io, puis envoie des packets UDP au port 6892 à de nombreuses adresses IP, celles-ci étant souvent séquentielles. De plus, certains auteurs de maliciels partagent leur code avec d'autres auteurs ou réutilisent le code d'un autre maliciel, ce qui rend les familles de maliciels interconnectées [20].

1.2 Importance de l'étiquetage

Chaque compagnie d'antivirus possède ses propres conventions et standards qu'elle utilise pour nommer les maliciels que leurs antivirus détectent. Pour un même maliciel, deux antivirus peuvent donner des noms différents. Virus Total [4] est un site qui permet de faire analyser un maliciel par de nombreux antivirus. En faisant analyser un maliciel donné par VirusTotal (voir tableau 1.1), il est possible de remarquer que certains antivirus comme Kingsoft (ligne 18) ne reconnaissent pas le fichier comme malicieux. Le tableau 1.1 montre aussi que chaque antivirus nomme le maliciel différemment, en y ajoutant des informations par exemple reliées au système d'exécution («*Win*64») ou encore au type de maliciel («*Trojan*», «*Rootkit*», «*Backdoor*»). De plus, certains antivirus classifient ce maliciel comme «*Kryptik*» tandis que le maliciel est en fait «*Necurs*», ce qui montre que les antivirus ne nomment pas toujours les maliciels de façon standardisée. En effet, l'objectif d'un antivirus est de reconnaître qu'un fichier

3. https://blog.malwarebytes.com/threat-analysis/2013/03/obfuscation-malwares-best-friend/
4. https://www.virustotal.com/

tableau 1.1 – Résultats d'analyses par des logiciels d'antivirus pour un maliciel donné

	Antivirus	**Résultat**
2	AVG	Win64/Cryptor
3	AegisLab	Uds.Dangerousobject.Multi|2|103 !c
4	AhnLab-V3	Trojan/Win64.Necurs.N1547679697
5	Avast	Win64 :Malware-gen
6	Avira	TR/Rootkit.Gen2
7	BitDefender	Trojan.Generic.14691383
8	Bkav	W32.FamVT.Rk64ND.Rootkit
9	CAT-QuickHeal	Trojan.Necurs.r6
10	Comodo	UnclassifiedMalware
11	CrowdStrike	malicious_confidence_69% (D)
12	Cyren	W64/Trojan.CRMH-1514
13	ESET-NOD32	a variant of Win64/Rootkit.Kryptik.AP
14	Fortinet	W64/Rootkit_Kryptik.AP !tr
15	Ikarus	Trojan.Win64.Rootkit
16	Invincea	trojan.win64.necurs.a
17	Kaspersky	UDS :DangerousObject.Multi.Generic
18	Kingsoft	
19	Malwarebytes	Rootkit.Necurs.GO
20	McAfee	Artemis !B4C59DCEF928
21	McAfee-GW-Edition	Artemis !Trojan
22	Microsoft	Trojan :Win64/Necurs.A
23	Rising	Malware.Generic !AEIpAqiZgXO@3 (thunder)
24	Sophos	Troj/Necurs-DC
25	Symantec	Backdoor.Necurs
26	Tencent	Win32.Trojan.Falsesign.Pgmn
27	TrendMicro	RTKT64_NECURS.YUV
28	ViRobot	Trojan.Win32.S.Agent.74688.A[h]
29	Yandex	Rootkit.Kryptik !fIcv5/MJQSM
30	Zillya	Rootkit.Kryptik.Win64.183

est malicieux et moins de nommer la menace avec précision.

Il est donc difficile d'extraire le nom de la famille d'un maliciel donné pour l'assigner comme étiquette au maliciel. Assigner une étiquette à un maliciel permet de regrouper plusieurs maliciels représentés par leur empreinte en un groupe et permet aussi aux analystes de maliciels d'étudier les maliciels de façon à ne pas analyser plu-

sieurs maliciels de la même famille (puisque ces derniers ont le même comportement). Pour assigner une étiquette à un maliciel donné, il faut donc effectuer un certain travail sur les résultats des antivirus pour en extraire les noms.

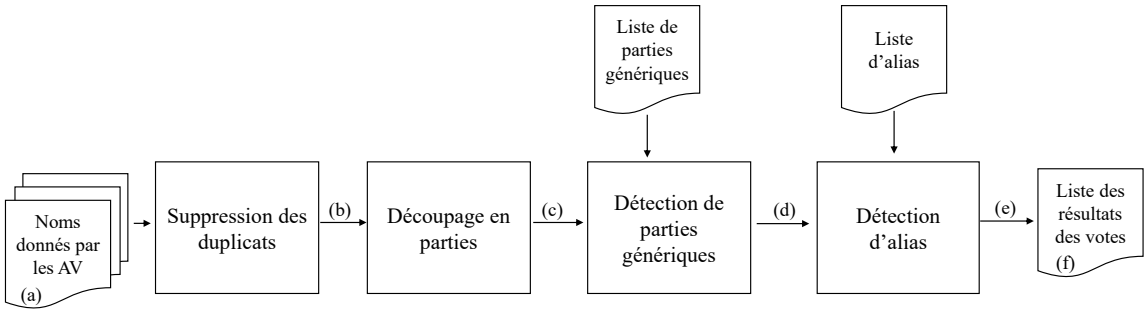


figure 1.2 – Architecture d'AVClass (Sebastián *& al.* [19]).

AVCLass L'outil AVClass développé par Sebastián *& al.* [19] permet d'extraire le nom d'un maliciel à l'aide d'un processus qui enlève les termes génériques du nom et qui élit par vote de pluralité le nom le plus probable pour le maliciel donné. Les étapes décrites dans les figures 1.2 et 1.3 sont les suivantes :

(a) La première phase consiste à prendre une liste de résultats d'antivirus et à filtrer les parties génériques.

(b) Tout d'abord, les duplicatas sont retirés de la liste, puis chaque élément de la liste est séparé selon un ensemble de caractères (« », «_», «-», *etc.*).

(c) Ensuite, les parties génériques comme par exemple «*Win*» ou «*Gen*», *etc.* du nom sont retirées.

(d) Lors de cette étape, les parties présentes dans une liste d'éléments à enlever sont retirées et les parties aléatoires ou qui comportent moins de quatre caractères sont enlevés.

(e) Ensuite, AVClass détecte et remplace les noms de variantes (*alias*) par le nom du maliciel original grâce à une liste de variantes pourvue en argument, ce qui permet de remplacer un nom par son nom de variante, le but étant de réduire le nombre de noms différents si ceux-ci réfèrent à la même famille.

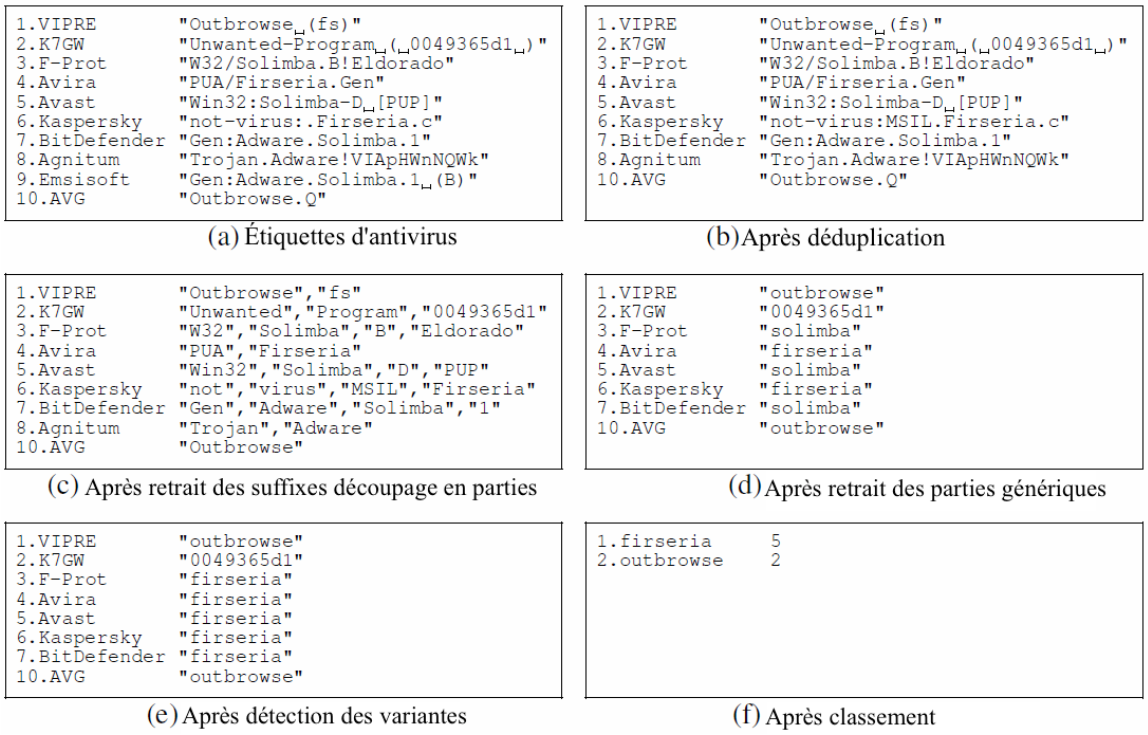


figure 1.3 – Exemple du processus d'AVClass [19].

(f) La deuxième phase prend les résultats de la première phase et organise cette liste du plus probable au moins probable selon un vote de pluralité, c'est-à-dire que plus un élément est présent dans la liste, plus ce nom est probable et doit être à une position élevée. Si deux noms obtiennent le même nombre de votes le plus élevés, alors un des deux noms est choisi selon l'ordre alphabétique décroissant. Les noms avec un nombre de votes inférieur à un seuil donné (par défaut, 1) ne peuvent pas être choisis comme nom le plus probable. Lorsqu'aucun nom le plus probable ne peut être élu, AVClass assigne le nom «SINGLETON» au maliciel, une catégorie qui veut dire qu'aucune étiquette valide n'a pu être trouvée grâce à la liste de résultats d'antivirus.

Le résultat d'AVClass est donc le nom qui a reçu le plus de votes par les antivirus, c'est-à-dire le nom plus probable selon les résultats des antivirus.

1.3 Processus de détection des maliciels

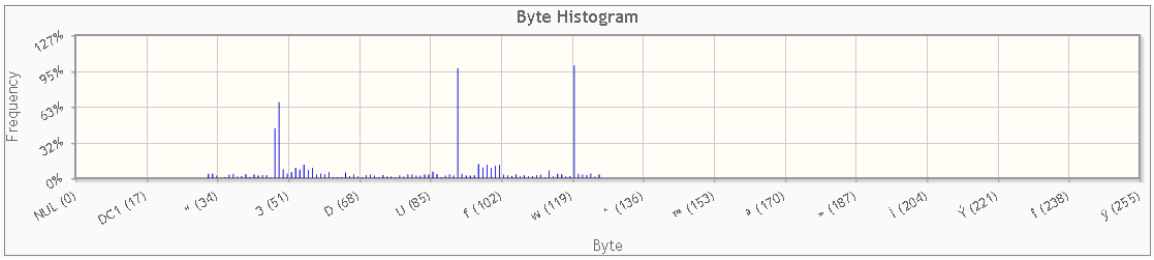
Lors de l'analyse d'un maliciel, il est possible de distinguer deux principaux types d'analyses différentes : *analyse statique* et *analyse dynamique*.

1.3.1 Analyse statique

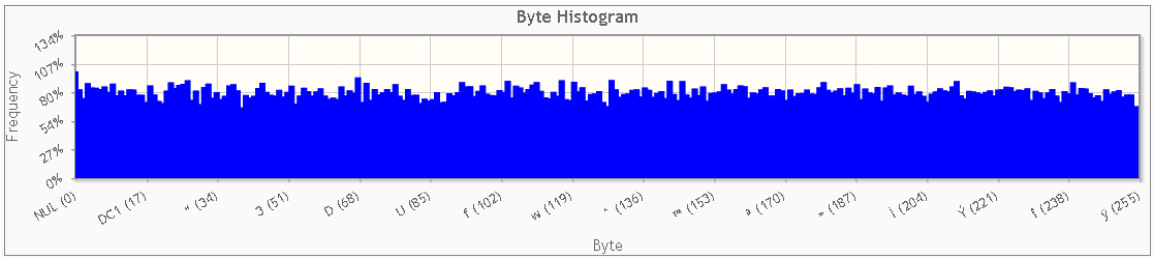
Elle est basée sur l'analyse du code [7]. Par exemple, il est possible d'extraire le type de fichier (e.g., EXE, DOC, PDF, TXT), les octets du fichiers, l'entropie. Ces informations recueillies du code source constituent les premières étapes pour pouvoir nommer et catégoriser un échantillon. En revanche, de nombreux échantillons utilisent des techniques d'obscurcissement et d'évasion comme par exemple l'utilisation d'obscurcisseurs pour compresser un fichier, rendant ainsi l'analyse statique du code inutile à moins que le fichier ait été décompressé à l'avance.

1.3.1.1 Entropie

L'entropie est une fonction mathématique utilisée pour évaluer la quantité d'information dans un fichier à partir de la fréquence des symboles que ce fichier contient (voir figure 1.4). Plus les symboles d'un fichier ont une fréquence similaire, plus son



(a) Exemple de fréquences de symboles d'un fichier non encrypté (entropie = 5.1)



(b) Exemple de fréquences de symboles d'un fichier encrypté (entropie = 8.0)

figure 1.4 – Exemple de fréquences de symboles de fichiers

entropie est grande. Ainsi un fichier encrypté dont le contenu est encodé de façon à ce que le contenu soit illisible pour quiconque ne possédant pas la clé permettant le déchiffrement, a une entropie élevée puisque tous les symboles qui composent le fichier ne forment pas des mots mais des suites de symboles illisibles. L'entropie est donc utile pour déterminer si un fichier est encrypté ou non, permettant de déterminer si un obscurcisseur a été utilisé pour cacher le contenu du fichier, ce qui est souvent utilisé par les maliciels.

1.3.1.2 Fichier *Portable Executable* (*PE*)

Un fichier exécutable possède un format particulier, qui dépend du système d'exécution. Le format d'un fichier exécutable sous Windows est appelé *Portable Executable* (*PE*), tandis que sous Linux un fichier exécutable est au format *Executable Linkable Format* (*ELF*) et sous Mac OS le format est *Mach Object*. Par exemple, un fichier *PE* est composé de plusieurs parties qui définissent différentes informations dont les tableaux d'éléments importés et exportés, des octets utilisés pour vérifier le type de fichier (*magic byte*) ou encore de l'information à propos des points d'entrée. Le tableau

1.2 illustre le format d'un fichier portable exécutable

tableau 1.2 – Format d'un fichier portable exécutable

1	En-tête MS-DOS
2	En-tête PE
3	En-tête optionnelle
4	Table des sections
5	Sections

Voici la description des différentes parties d'un fichier *PE*[5].

(1) En-tête MS-DOS – La première partie d'un fichier *PE* est l'en-tête DOS. Cette section contient toujours les lettres «MZ» et est suivie plus loin par la chaîne de caractères *«This program cannot be run in DOS mode»* (voir figure 1.5). Cette chaîne de caractère est utilisée pour assurer la rétrocompatibilité avec les systèmes DOS 16-bits, de façon à ce que sur ces systèmes, un message d'erreur lisible soit affiché.

(2) En-tête PE – La seconde partie d'un fichier *PE* est l'en-tête *PE*. Cette partie contient les lettres «PE» (voir figure 1.5) et assure que ce fichier est bien portable exécutable.

(3) En-tête optionnelle – Cette partie d'un fichier portable exécutable contient différentes informations importantes telles que la table des imports, des exports, une table de ressources et l'adresse du point d'entrée du programme.

5. Pour plus d'information sur le contenu des sections d'une en-tête d'un fichier PE, voir https://en.wikipedia.org/wiki/Portable_Executable#/media/File:Portable_Executable_32_bit_Structure_in_SVG_fixed.svg ou encore http://brokenthorn.com/Resources/OSDevPE.html

```
0000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ..............
0000010: b800 0000 0000 0000 4000 0000 0000 0000  ........@.......
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000030: 0000 0000 0000 0000 0000 0000 f800 0000  ................
0000040: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  ........!..L.!Th
0000050: 6973 2070 726f 6772 616d 2063 616e 6e6f  is program canno
0000060: 7420 6265 2072 756e 2069 6e20 444f 5320  t be run in DOS
0000070: 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000  mode....$.......
0000080: c21e 94bf 867f faec 867f faec 867f faec  ................
0000090: 1531 62ec 847f faec 9de2 50ec 297f faec  .1b.......P.)...
00000a0: 9de2 51ec b37f faec 8f07 79ec 8f7f faec  ..Q.......y.....
00000b0: 8f07 69ec a77f faec 867f fbec 967d faec  ..i..........}..
00000c0: 9de2 4eec ce7f faec 9de2 64ec 9a7f faec  ..N.......d.....
00000d0: 9de2 60ec 877f faec 867f 6dec 877f faec  ..`.......m.....
00000e0: 9de2 67ec 877f faec 5269 6368 867f faec  ..g.....Rich....
00000f0: 0000 0000 0000 0000 5045 0000 4c01 0400  ........PE..L...
```

figure 1.5 – Premiers octets d'un fichier portable exécutable

(4) Table des sections – Cette partie contient le nombre de sections ainsi qu'un résumé de chaque section du fichier *PE*, comme par exemple sa taille ou son nom. Dans cette partie sont aussi présentes les tables d'imports et d'exports. Les tables d'imports listent toutes les bibliothèques qui peuvent être utilisées par le programme exécutable et toutes les méthodes présentes dans ces bibliothèques qui pourraient être appelées par le programme.

(5) Sections – Cette partie contient le contenu du programme ainsi que le point d'entrée du programme. Certaines sections sont prédéfinies comme par exemple la section du code exécutable, la section des données ou encore la section des ressources.

1.3.2 Analyse dynamique

Elle constitue à lancer l'exécution du maliciel dans un environnement contrôlé et à observer ses actions [7]. Cette technique est plus résistante à l'obscurcissement du code puisque l'analyse porte sur son comportement visible au lieu du code obscur. Plusieurs types de comportements peuvent être enregistrés lors de l'analyse dynamique, comme par exemple les actions du maliciel sur l'hôte ou encore les actions de ce dernier sur

le réseau. L'analyse dynamique dépend cependant de la façon dont le maliciel est exécuté et de l'environnement dans lequel ce dernier est exécuté.

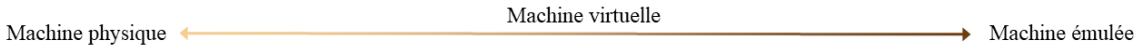
La plupart des analyses dynamiques ont une limite temporelle, par exemple une minute, dix minutes. En effet, il est impossible de savoir combien de temps un maliciel est actif et effectue des actions, c'est pourquoi la plupart des analyses dynamiques exécutent les maliciels pour un temps donné. De plus, certains maliciels ne démarrent pas leurs activités dès le début de leur exécution mais attendent un certain délai ou une date précise [3], ce qui leur permet d'éviter d'être détectés par des analyses dynamiques automatiques ou encore de rendre plus difficile la détermination de la source de l'infection (voir section 1.1.1.1). Certains maliciels attendent même qu'un téléchargement soit fait pour commencer leur exécution, ce qui donne l'impression que le dernier fichier téléchargé est la source de l'infection. D'autre part, l'exécution d'un maliciel peut être plus longue que la limite de temps de l'analyse dynamique, ce qui résulte en une trace non complète d'exécution retournée par l'analyse dynamique d'un maliciel.

L'environnement dans lequel un maliciel est exécuté a aussi une influence sur la trace d'exécution retournée par l'analyse dynamique pour un maliciel. En effet, l'environnement d'une analyse dynamique doit être contrôlé et reproductible, pour éviter que le maliciel infecte d'autres systèmes et pour permettre d'avoir toujours les mêmes conditions d'environnement au début de l'analyse.

Bac à sable (*sandbox* en anglais) La plupart des analyses dynamiques sont effectuées dans un bac à sable. Un bac à sable est un environnement contrôlé qui est utilisé pour reproduire un environnement réel. Les caractéristiques et composantes de ce bac à sable sont connues, ce qui permet d'être capable de reproduire le bac à sable à volonté.

Une caractéristique du bac à sable est la façon dont il est connecté au réseau. En effet, il est possible d'isoler complètement un bac à sable pour qu'il ait accès à un réseau complètement émulé ou tout au contraire de le laisser accéder à internet sans restrictions (voir figure 1.6a). Entre ces deux extrêmes, on peut trouver des bacs à sable qui permettent d'accéder à internet avec certaines restrictions ou encore des bacs à sables qui n'ont pas accès à internet. Certaines technologies comme par exemple

(a) Échelles de caractéristiques d'un bac à sable au niveau du réseau



(b) Échelles de caractéristiques d'un bac à sable au niveau de la virtualisation

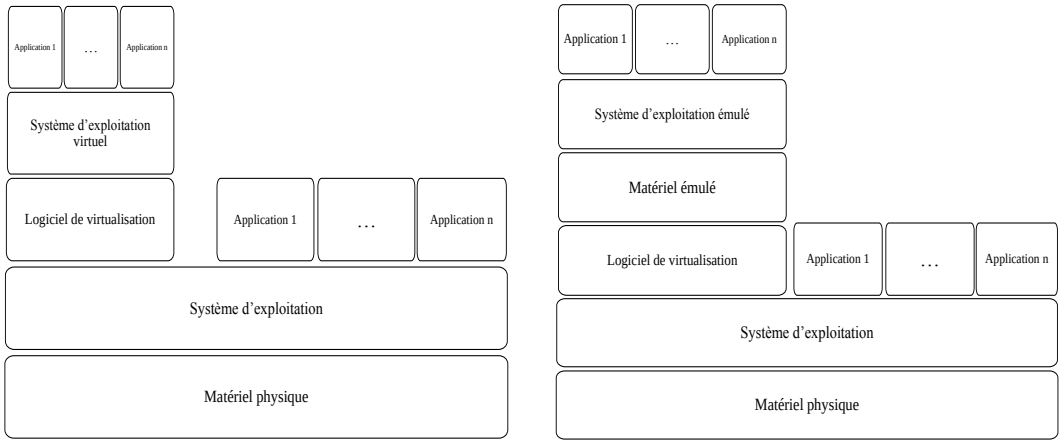
figure 1.6 – Échelles de caractéristiques d'un bac à sable

InetSim [6] ou encore Cuckoo [7] peuvent être utilisées pour émuler un réseau internet.

Un bac à sable peut habituellement opérer sur différents environnements : directement sur la machine physique, sur une machine virtuelle ou encore sur une machine émulée (voir figure 1.6b). Une machine virtuelle simule des parties du matériel physique d'une machine à l'aide d'opérations effectuées par le logiciel, tandis que le reste des opérations sont effectuées directement sur le matériel physique de la machine (voir figure 1.7a). Lors de virtualisation, le système d'exploitation de la machine physique est utilisé pour exécuter certaines instructions, ce qui veut dire que le système d'exploitation utilisé par la machine virtuelle doit être compatible avec l'architecture de la machine physique. L'émulation d'une machine consiste à simuler la partie physique de la machine avec un logiciel (voir figure 1.7b). Avec l'émulation, le système d'exploitation de la machine émulée n'a pas nécessairement besoin d'être compatible avec l'architecture de la machine physique.

Puisqu'un bac à sable est contrôlé et reproductible, il permet d'exécuter un maliciel inconnu à l'intérieur de l'environnement sans affecter le reste des environnements. En effet, un bac à sable est utile pour s'assurer que les conditions de test sont toujours les mêmes d'un maliciel à l'autre, puisque l'environnement est isolé et qu'il est remis à son état initial entre chaque maliciel exécuté. Les conditions initiales et caractéristiques du bac à sable étant connues, il est possible de déterminer quels changements ont été effectués par le logiciel exécuté. De plus, l'utilisation d'un environnement contrôlé permet l'enregistrement des actions (réseau, mémoire, actions sur le système d'exploitation, actions sur les fichiers, *etc.*) faites par le logiciel.

6. http://www.inetsim.org/
7. https://cuckoosandbox.org/



(a) Exemple d'un bac à sable virtuel (b) Exemple d'un bac à sable émulé

figure 1.7 – Exemples de bacs à sable

1.4 Regroupement

Le regroupement (*clustering* en anglais) consiste à regrouper des éléments qui se ressemblent et qui sont proches les uns des autres. En autres mots, le regroupement consiste à organiser des données de sorte que la similarité des éléments au sein d'un même groupe soit forte, tandis que la similarité entre les éléments de groupes différents soit faible.

Regroupement de maliciels Le principe de pouvoir regrouper efficacement des éléments auparavant inconnus en groupes est très utile pour l'analyse de maliciels, tout d'abord dû au nombre important de nouveaux maliciels recensés tous les jours et dû au fait que la plupart de ces derniers sont des variations de maliciels déjà connus. S'il est possible de regrouper des maliciels de façon automatique, les analystes peuvent alors analyser manuellement les maliciels de façon éclairée. En effet, plutôt que d'analyser un maliciel qui se révèle après analyse être un maliciel d'une famille connue et déjà analysée, les analystes peuvent se concentrer sur les maliciels inconnus ou qui ne font partie d'aucun groupe.

1.4.1 Algorithmes de regroupement

1.4.1.1 Regroupement hiérarchique

Le regroupement hiérarchique (*Connectivity-based clustering*) consiste à créer une hiérarchie de groupes par le regroupement successif. Il existe plusieurs mesures de similarité utilisées par le regroupement hiérarchique. Ces mesures sont utilisées pour déterminer les groupes à fusionner.

Liaison simple (*Single linkage*) (*nearest neighbour clustering*) – La liaison simple de la distance entre deux groupes correspond à la distance minimale des membres du groupe à tout autre membre d'un autre groupe.

Liaison moyenne (*Average linkage*) – La liaison moyenne de la distance entre deux groupes est égale à la distance moyenne des membres du groupe à tout autre membre d'un autre groupe.

Liaison complète (*Complete linkage*) (*farthest neighbour clustering*) – La liaison complète de la distance entre deux groupes correspond à la distance maximale des membres du groupe à tout autre membre d'un autre groupe.

1.4.1.2 Regroupement par partition

Il existe plusieurs méthodes de regroupement par partition, dont le but est de partitionner l'espace, où chaque partition équivaut à un groupe.

Regroupement par centroides (*Centroid-based clustering*) – Le regroupement par centroides consiste à représenter chaque groupe par un vecteur central et à séparer les éléments en un nombre donné de partitions. L'algorithme effectue plusieurs rondes de regroupement. Durant chaque ronde, tous les éléments sont placés dans des groupes en fonction de leur distance par rapport aux vecteurs centraux. Lors de la première ronde, les vecteurs centraux sont initialisés de façon aléatoire, puis à chaque ronde subséquente, les centres sont calculés en faisant la moyenne des membres de chaque groupe. L'algorithme s'arrête lorsque les centres ne changent pas ou que la valeur du changement est en dessous d'un seuil donné. L'algorithme k-moyennes [8] est un exemple de ce type de regroupe-

8. https ://fr.wikipedia.org/wiki/K-moyennes

ment.

Regroupement par distribution (*Distribution-based clustering*) – Le regroupement par distribution consiste à regrouper les éléments grâce à un modèle statistique de distribution. Les groupes sont définis en fonction de la probabilité que des éléments d'un même groupe appartiennent à un même groupe. Le mélange de Gaussiens déterminé par l'algorithme *espérance-maximisation*[9] est un exemple de ce type de regroupement.

Regroupement par densité (*Density-based clustering*) – Le regroupement par densité est basé sur la densité des groupes. Les groupes sont définis comme des aires où la densité à l'intérieur de l'aire est plus importante que la densité du reste de l'espace de données. L'algorithme DBSCAN[10] est un exemple de ce type de regroupement.

Chaque algorithme de regroupement peut être classé selon plusieurs caractéristiques [10].

Ascendant ou descendant – Un algorithme ascendant commence par considérer chaque élément comme un groupe. Ensuite, les groupes les plus proches selon une mesure de distance donnée sont fusionnés ensembles. Le regroupement s'arrête lorsque la distance minimale entre les groupes atteint un certain seuil ou que tous les éléments sont dans un même groupe. Un algorithme descendant commence en considérant tous les éléments comme un seul groupe. Ensuite, le groupe est séparé en groupes plus petits et ce jusqu'à ce que la distance maximale entre chaque groupe atteigne un certain seuil ou que chaque élément constitue un groupe.

Monothétique ou polythétique – Un algorithme de regroupement est monothétique si le calcul de la distance s'effectue grâce à une seule des caractéristiques des vecteurs. Au contraire, un algorithme de regroupement est polythétique si toutes les caractéristiques des vecteurs sont prises en compte pour le calcul de la distance.

Précis ou flou : Avec un algorithme de regroupement qui est précis, un point ne peut

9. https ://fr.wikipedia.org/wiki/Algorithme_espérance-maximisation
10. https ://fr.wikipedia.org/wiki/DBSCAN

appartenir qu'à un seul groupe, tandis qu'avec un algorithme de regroupement flou, un point peut appartenir à un ou plusieurs groupes.

Incrémental ou non incrémental – Un algorithme de regroupement est incrémental s'il permet de mettre à jour les groupes au fur et à mesure du temps, tandis qu'un algorithme non incrémental effectue le regroupement complet à chaque exécution.

1.4.1.3 Vecteurs et mesures de distance

Une partie importante d'un algorithme de regroupement est la façon dont la distance entre les points est calculée. Chaque maliciel est représenté par un vecteur de caractéristiques. Un vecteur de caractéristiques (*feature vector*) est un vecteur de m dimensions qui représente un objet. Les caractéristiques de chaque vecteur sont des propriétés mesurables de l'objet qui le définissent. Dans ce mémoire, chaque maliciel est représenté par un vecteur de caractéristiques de taille m et les caractéristiques correspondent aux n-grammes extraits du comportement du maliciel.

Dans un cadre général, un n-gramme est une combinaison de n symboles, parties ou actions successives divisées par une limite donnée. Par exemple, il est possible d'extraire tous les bi-grammes de mots possibles d'une phrase donnée : «*Lorem ipsum dolor sit amet*». Les bi-grammes extraits de cette phrase sont «*Lorem ipsum*», «*ipsum dolor*», «*dolor sit*» et «*sit amet*». Dans ce mémoire, un n-gramme est une combinaison de n actions du maliciel (action sur l'hôte, import de bibliothèque ou action réseau) successives, la limite d'un n étant la fin d'une action. Ceci permet de prendre en compte l'ordre dans lequel les actions d'un maliciel ont été effectuées et de pouvoir étudier la similarité entre deux maliciels par rapport à des séquences d'actions.

Normalisation La norme est souvent utilisée pour permettre de mieux comparer et de normaliser les caractéristiques des vecteurs. Pour cela, la norme du vecteur de caractéristiques est calculée grâce à la norme euclidienne (L2) suivante :

$$\text{Norme L2} = \sqrt{\sum_{i=1}^{m} x_i^2},$$

tableau 1.3 – Exemple de fréquences de termes

bi-gramme	a	b
`Ceci est`	1	0
`est un`	1	0
`Voici un`	0	1
`un exemple`	1	1

avec x une caractéristique d'un vecteur de caractéristiques de taille m. Chaque caractéristique des vecteurs est ensuite divisée par la norme L2 pour en uniformiser les valeurs.

Distances et dissimilitude La distance euclidienne est la distance d'un segment connectant vecteurs de caractéristiques donnés :

$$\sqrt{\sum_{i=1}^{m}(q_i - p_i)^2},$$

avec q et p des vecteurs de caractéristiques de taille m.

La similarité cosinus est souvent utilisée dans du regroupement de textes pour déterminer la distance entre deux documents. Celle-ci détermine l'angle entre deux vecteurs et se calcule grâce à la formule suivante :

$$S_{qp} = \frac{q \cdot p}{\|q\|\|p\|},$$

avec q et p des vecteurs de caractéristiques de taille m.

Pour pouvoir utiliser la similarité cosinus avec des chaînes de caractères, la fréquence de chaque terme ou n-gramme est calculée, résultant ainsi en un vecteur de fréquence. Ce vecteur peut ensuite être utilisé pour calculer la similarité cosinus. Un exemple de vecteur de fréquences pour les chaînes de caractères *«Ceci est un exemple»* et *«Voici un exemple»* est montré dans le tableau 1.3, où la fréquence des bi-grammes peut être représentée par les vecteurs a et b. La dissimilitude cosinus peut être dérivée de la similarité *cosinus* en utilisant la formule $1 - S_{qp}$.

Dans l'exemple du tableau 1.3, la similarité cosinus est calculée comme suit :

$$S_{ab} = \frac{a \cdot b}{\|a\|\|b\|} = \frac{(1,1,0,1) \cdot (0,0,1,1)}{\sqrt{1^2+1^2+0^2+1^2}\sqrt{0^2+0^2+1^2+1^2}} = \frac{1}{\sqrt{3}\sqrt{2}} \approx 0,58.$$

La similarité cosinus entre *«Ceci est un exemple»* et *«Voici un exemple»* est approximativement de 0,58, c'est-à-dire que les deux chaînes de caractères sont relativement similaires. La dissimilitude cosinus pour cet exemple est de $1 - 0,58 = 0,42$.

1.4.2 Évaluation du regroupement

Deux façons d'évaluer les graphes résultant d'algorithmes de regroupement sont étudiées : l'évaluation externe et l'évaluation interne.

1.4.2.1 Évaluation externe

L'évaluation externe se base sur des valeurs de références, où chaque élément utilisé en entrée dans l'algorithme possède une étiquette qui permet de le classer. Il est ainsi possible de vérifier si le résultat de l'algorithme de regroupement est proche ou non des classes d'étiquettes prédéterminées. Les étiquettes représentent ainsi une vérité et ces dernières sont supposées justes.
Pour avoir des étiquettes nous permettant d'effectuer une évaluation externe, nous avons fait analyser chaque maliciel par un nombre d'antivirus. Chaque antivirus décide d'une étiquette selon sa convention de nommage, comme par exemple « *Trojan.FakeAlert*». Pour uniformiser les résultats des antivirus et avoir une étiquette par maliciel, nous avons utilisé AVClass qui permet d'extraire l'étiquette la plus probable pour chaque maliciel (voir 1.2).

Plusieurs mesures permettent d'effectuer une évaluation externe, dont les matrices de confusion. Une matrice de confusion est un outil qui permet de mesurer la qualité du regroupement effectué en comparant les données de référence aux données utilisées pour la classification, ce qui permet d'obtenir le nombre d'éléments correctement prédits. Une matrice de confusion résultante d'un regroupement permet de comparer un algorithme de regroupement à un autre.

Grâce à la matrice de confusion, il est possible de rassembler :

— **vrais positifs (VP)** – éléments de la classe X étudiée, correctement placés dans la classe X ;
— **vrais négatifs (VN)** – éléments d'une autre classe $\neg X$, correctement placés dans la classe $\neg X$;
— **faux positifs (FP)** – éléments d'une autre classe $\neg X$, placés dans la classe X ;
— **faux négatifs (FN)** – éléments de la classe X étudiée, placés dans une autre classe $\neg X$.

Par exemple, le tableau 1.4 montre une matrice de confusion pour sept classes de maliciels différents : «*andromeda*», «*asprox*», «*bedep*», «*blackshades*», «*bozok*», «*cerber*» et «*citadel*». Dans ce tableau, il est possible de remarquer que les maliciels de la famille «*cerber*» ont été correctement classifiés lors du regroupement. En effet, sur les dix-huit maliciels pour lesquels le nom était marqué «*cerber*» dans la référence, dix-huit ont été prédits comme «*cerber*». En revanche, les maliciels de la famille «*andromeda*» ont été moins bien classifiés. En effet, sur les 1115 maliciels que la référence marquait comme «*andromeda*», seuls 1038 ont été bien classifiés.

tableau 1.4 – Exemple de matrice de confusion

↓ Prédiction \ Référence →	andromeda	asprox	bedep	blackshades	bozok	cerber	citadel
andromeda	1038	0	0	0	0	0	0
asprox	13	1052	0	0	0	0	0
bedep	38	3	629	0	0	0	0
blackshades	1	0	0	1	0	0	0
bozok	10	0	1	1	69	0	0
cerber	0	0	0	0	0	18	0
citadel	15	0	1	0	0	0	746

Définition 1.4.1 (Exactitude (*accuracy*)) *L'exactitude montre la correctitude du classifieur, c'est-à-dire combien de fois il est correct. La formule pour calculer l'exactitude est la suivante :*

$$Exactitude = \frac{VP+VN}{VP+VN+FP+FN}$$

Définition 1.4.2 (Précision (*precision*)) *La précision est la proportion d'éléments bien classés. Une haute précision montre que la proportion de résultats pertinents est élevée, c'est-à-dire que la proportion de vrais positifs est élevée. Une basse précision indique au contraire un nombre important de faux positifs.*

$$Precision = \frac{VP}{VP+FP}$$

Définition 1.4.3 (Rappel (*recall*)) *Le rappel est la proportion d'éléments bien classés par rapport au nombre d'éléments de la classe à prédire. Un haut rappel montre que l'algorithme a retourné une proportion importante d'éléments pertinents, c'est-à-dire de vrais positifs, parmi l'ensemble des éléments appartenant à la classe à prédire. Un rappel bas signifie un nombre important de faux négatifs :*

$$Rappel = \frac{VP}{VP+FN}$$

Définition 1.4.4 (F-Mesure) *La F-Mesure est la moyenne harmonique de la précision et du rappel, elle se calcule de la façon suivante :*

$$F\text{-}Mesure = \frac{précision \times rappel}{précision + rappel}$$

Définition 1.4.5 (Coefficient de Jaccard) *Le coefficient de Jaccard calcule la similarité entre deux jeux de données. Un coefficient de Jaccard de 1 montre que les deux jeux de données sont égaux, tandis qu'un coefficient de Jaccard de 0 montre qu'ils n'ont aucun éléments en commun :*

$$Coefficient\ de\ Jaccard = \frac{VP}{VP+FP+FN}$$

1.4.2.2 Évaluation interne

L'évaluation interne s'effectue essentiellement en fonction des similitudes entre les éléments de chaque groupe (variance intragroupe) et les dissimilitudes entre les groupes (variance intergroupe). Plus le graphe possède de fortes similarités intragroupes et peu de similarités intergroupes, plus le score de l'évaluation interne est élevé. Cette évaluation est utile pour déterminer la performance d'un algorithme par rapport à un autre. Des exemples de mesures d'évaluation interne sont l'indice Davies-Bouldin ou encore le coefficient Silhouette.

L'indice de Davies-Bouldin évalue la similarité intragroupe et les différences intergroupes. Le but est de déterminer la similarité d'un élément par rapport aux autres éléments de son groupe et la dissimilitude entre les groupes en calculant la distance

entre les centres. Une valeur proche de 0 indique qu'un élément est bien placé. L'indice de Davies-Bouldin, noté DB, est défini comme suit :

$$DB = \frac{1}{n}\sum_{i=1}^{n}\max_{j\neq i}\left\{\frac{\sigma_i + \sigma_j}{d(c_i, c_j)}\right\}$$

avec n le nombre de groupes, c_x le centre ou prototype du groupe x, σ_x la distance moyenne des éléments du groupe x à son centre (c_x), et $d(c_i, c_j)$ la distance entre les centres c_i et c_j.

L'indice Silhouette mesure la distance entre chaque point et son centre et le centre des autres groupes. Le but est de déterminer la similarité d'un élément par rapport aux autres éléments de son groupe comparé à la similarité de l'élément par rapport aux éléments des autres groupes. L'indice Silhouette peut prendre des valeurs entre -1 et 1, sachant qu'une valeur plus grande indique qu'un élément est bien placé.

Soit N le nombre de groupes, n_x le nombre d'éléments dans un groupe x et E_i^x l'élément i appartenant au groupe x. L'indice Silhouette pour un groupe x se calcule de la façon suivante :

$$S_x = \frac{1}{n_x}\sum_{i=1}^{n_x}\left(\frac{b_i^x - a_i^x}{\max\left\{a_i^x, b_i^x\right\}}\right),$$

avec a_i^x la distance moyenne d'un élément i appartenant au groupe x aux autres éléments du groupe x et b_i^x la distance minimale des distances moyennes de i à tous les autres éléments des autres groupes dont i n'appartient pas. Il est possible de calculer a_i^x et b_i^x grâce aux formules suivantes :

$$a_i^x = \frac{1}{n_x - 1}\sum_{\substack{j=1\\ j\neq i}}^{n_x} d(E_i^x, E_j^x) \quad \text{et} \quad b_i^x = \min_{\substack{y=1,\ldots,N\\ y\neq x}}\left\{\frac{1}{n_y}\sum_{\substack{j=1\\ j\neq i}}^{n_y} d(E_i^x, E_j^y)\right\}.$$

Enfin, l'indice Silhouette global se calcule comme suit :

$$S = \frac{1}{N}\sum_{x=1}^{N} S_x.$$

L'analyse externe permet de vérifier la validité de la classification en se basant sur des valeurs de références connues pour être vraies. Ce type d'évaluation est complé-

mentaire avec l'analyse interne qui ne vérifie pas la validité de la classification, mais seulement la cohésion des groupes.

Conclusion

Les concepts décrits dans ce chapitre forment la base de connaissances nécessaires à la lecture de ce mémoire. Dans ce chapitre, nous avons vu des concepts propres à la cybersécurité, des techniques d'analyse et de détection de maliciels et le regroupement en général.

Chapitre 2

Revue de littérature sur le regroupement de maliciels

Après avoir présenté les concepts de base du regroupement et de la cybersécurité, nous pouvons désormais lier ces concepts ensemble pour étudier l'état de la recherche sur le regroupement de maliciels et des données utilisées. De plus, nous examinons dans ce chapitre où en est l'état de l'art à propos de l'évolution du regroupement, de façon à pouvoir l'incorporer dans un système de regroupement en temps réel.

Nous commençons par exposer les différentes techniques utilisées dans le regroupement dans la section 2.1 en décrivant les différentes mesures de distance, de qualité, les données utilisées mais aussi en montrant quels algorithmes sont utilisés pour effectuer le regroupement de maliciels. Nous étudions par la suite le regroupement à travers le temps (évolutif). Ensuite, nous nous intéressons à l'extraction de signature dans la section 2.2 puis terminons ce chapitre par l'étude de l'étiquetage de données dans la section 2.3.

2.1 Regroupement

Comme présenté dans le chapitre 1, le regroupement est une technique qui s'applique à de nombreux domaines comme la biologie, la psychologie et la géographie. Le regroupement peut être utilisé pour explorer les données ou pour confirmer un phéno-

mène ou une tendance. Il est en effet utilisé pour permettre de regrouper des données en groupes selon une ou plusieurs caractéristiques précises. Selon le domaine et les données à regrouper, l'algorithme, les mesures et traitements de données à appliquer diffèrent [10]. Comme montré précédemment, il existe plusieurs types de regroupements (voir section 1.4) et caractéristiques et ces dernières sont utilisées pour classer des algorithmes de regroupement proposés dans la littérature. Dans cette section, nous concentrons surtout notre attention sur les auteurs traitant du regroupement de maliciels.

2.1.1 Mesures de distance

Chaque algorithme de regroupement utilise une mesure de distance pour calculer la similitude entre les différents éléments à regrouper. La distance euclidienne est un cas particulier de la mesure de Minkowski [1] et est d'habitude utilisée pour évaluer la distance entre des objets. Cependant, cette distance est sensible aux données éparses, c'est pourquoi certains auteurs utilisent d'autres types de distances comme la distance de Mahalanobis [10] ou encore une distance appelée Normalized Compression Distance [2], qui est une mesure de distance entre deux objets. La similitude cosinus (voir section 1.4.1.3) est une métrique souvent utilisée dans la fouille de documents ou avec des chaînes de caractères et n'est pas sensible aux données éparses. Cette métrique est souvent couplée à un calcul de fréquence des termes (TF-IDF) pour construire les vecteurs utilisés lors du calcul de similitude cosinus. Chaque mesure de distance choisie dépend des données et les résultats d'un algorithme de regroupement dépendent grandement de la mesure de distance choisie. Il est aussi possible d'utiliser la normalisation (voir sous-section 1.4.1.3) pour uniformiser les caractéristiques et ainsi permettre un regroupement plus précis.

2.1.2 Mesures de qualité

Il est important de noter que le regroupement est aussi appelé apprentissage non-supervisé, ce qui veut dire qu'il n'y a pas d'étiquette, comme par exemple le numéro ou nom de classe, associée à chaque donnée à classifier [10]. Des mesures d'évaluation

1. https://en.wikipedia.org/wiki/Minkowski_distance

du regroupement (voir sous-section 1.4.2) existent pour déterminer la validité et l'efficacité du regroupement, qui est un sujet important. Certains auteurs utilisent des mesures d'évaluation externes, comme par exemple la précision et le rappel [3], tandis que d'autres utilisent des mesures d'évaluation internes comme par exemple Davies-Bouldin [14]. Petrovic [15] montre de plus que Silhouette produit des résultats plus précis, mais le temps de calcul est plus long pour Silhouette que pour Davies-Bouldin.

2.1.3 Données

Schultz *& al.* [18] ont été les premiers à introduire le concept de regroupement pour le domaine de la classification de maliciels. Dans leur étude, ils utilisent des résultats d'analyses statiques comme données pour le regroupement. Ils extraient de fichiers *Portable Executable* (voir sous-section 1.3.1.2) la liste des bibliothèques utilisées, la liste des méthodes appelées dans ces bibliothèques ou encore le nombre d'appels systèmes. Ils utilisent aussi les chaînes de caractères et des séquences d'octets (*n*-grammes, voir sous-section 1.4.1.3) contenus dans des fichiers exécutables. Hu et Tan [8] avancent que l'extraction de n-grammes à partir d'octets de fichiers malicieux est une technique de classification qui donne habituellement de bon résultats, cependant, cette technique est coûteuse en temps de calcul. Hu et Tan [8] proposent une technique de sélection des n-grammes grâce à une «fenêtre coulissante» de taille n se déplaçant du début à la fin du fichier pour chaque fichier exécutable. À chaque étape de la fenêtre correspond à un n-gramme pour procéder à leur extraction. Les duplicatas sont ensuite enlevés et l'empreinte (voir sous-section 1.1.1.4) de chaque n-gramme est calculée. D'après Hu et Tan [8], une valeur de $n = 4$ pour des n-grammes est habituellement une valeur qui permet de classifier de façon robuste sans perdre en performance.

Cependant, les représentations issues d'analyses statiques se basent sur le code du maliciel, qui peut facilement être caché ou modifié des mécanismes de contournement de détection des maliciels [1, 7, 11, 12, 20, 22, 23] (voir sous-section 1.1.2.2). C'est pourquoi de nombreux auteurs se tournent vers des représentations issues de l'analyse dynamique pour effectuer des regroupements de maliciels.

Certains auteurs se concentrent sur l'aspect réseau du comportement d'un maliciel

et collectent les traces réseaux enregistrées lors d'analyses dynamiques. Par exemple, Perdisci *& al.* [14] présentent une technique basée sur des enregistrements de traces réseaux effectués lors d'analyses dynamiques. Nari et Ghorbani [11] utilisent aussi des traces réseaux pour leur classification et forment un graphe de comportement pour représenter les activités réseau du maliciel. Rafique et Caballero [16] font exécuter les échantillons dans un environnement virtuel (bac à sable, voir section 1.3.2), puis collectent les traces réseaux résultantes qui sont ensuite combinées avec des traces réseaux de traffic bénin pour être assemblées en groupes.

D'autres auteurs se concentrent sur les actions du maliciel sur l'hôte (voir sous-section 1.1.1.1) pour effectuer des regroupements. Rieck *& al.* [17] proposent un outil qui utilise des données résultant d'exécution de maliciels dans un bac à sable. Bayer *& al.* [3] utilisent des enregistrements d'actions effectuées sur l'hôte comme données de regroupement. Bailey *& al.* [2] montrent également une technique de classification en termes de changements de l'état de l'hôte. Ils effectuent l'exécution d'analyse dynamique dans un bac à sable virtuel avec pare-feu partiel et utilisent les enregistrements collectés comme données dans leur algorithme. L'utilisation de traces réseaux ou de données provenant des actions du maliciel sur l'hôte permet une plus grande résistance au métamorphisme et à l'obscurcissement des maliciels.

Certains auteurs tentent de combiner des représentations provenant d'analyses statiques et dynamiques pour permettre un meilleur regroupement. Islam *& al.* [9] combinent trois caractéristiques pour la classification de nouveaux maliciels en deux groupes (malicieux, bénins) ; deux caractéristiques sont extraites d'analyses statiques et une autre extraite d'analyses dynamiques. Ils utilisent la fréquence des longueurs des méthodes, les chaînes de caractères extraites d'analyses statiques, les noms de bibliothèques et les paramètres utilisés lors des actions sur l'hôte extraits de l'analyse dynamique. Ils trouvent qu'en combinant des résultats d'analyses statiques et dynamiques, les résultats sont supérieurs aux autres travaux ne prenant en compte qu'un des deux aspects.

2.1.4 Technique de regroupement

2.1.4.1 Regroupement de maliciels

Bayer *& al.* [3] utilisent une technique de regroupement basée sur *Locality Sensitive Hashing (LSH)* qui s'inspire de l'approche du plus proche voisin. Cette technique est utilisée pour calculer un petit nombre de distances au lieu des $\frac{n^2}{2}$ distances en temps normal, ce qui permet de regrouper un grand nombre d'éléments en un temps raisonnable (75 000 maliciels en 3 heures). Ils utilisent un algorithme hiérarchique ascendant avec une approche de liaison simple pour regrouper leurs données. Bailey *& al.* [2] montrent eux aussi un algorithme de regroupement hiérarchique utilisant un fusionnement à liaison simple pour regrouper les traces d'exécution collectées. Leur algorithme est polythétique, précis et non incrémental. FIRMA [16] est un partitionnement descendant, polythétique, précis et non incrémental et regroupe les maliciels en groupes de familles de maliciels. Cet algorithme utilisé par Rafique et Caballero [16] parvient à créer un petit nombre de groupes de familles, même lorsque le nombre initial de groupes de traffic réseau est grand. Perdisci *& al.* [13] utilisent un algorithme hiérarchique à liaison simple, descendant, polythétique, précis et non incrémental. La première étape de cet algorithme est un regroupement grossier suivi d'un regroupement plus raffiné pour chaque groupe créé lors du premier regroupement. Cette technique permet de réduire le temps de calcul du regroupement. Ensuite vient une étape de fusion des groupes considérés trop spécifiques pour les rendre plus génériques. Perdisci *& al.* [13] améliorent cet algorithme pour le rendre plus performant lors du regroupement sur de larges ensembles de données.

Rieck *& al.* [17] proposent un outil qui utilise des données résultant d'une exécution du maliciel dans un bac à sable pour le regroupement effectué par l'algorithme qu'ils proposent : Malheur. Malheur est un algorithme hiérarchique ascendant, polythétique, précis et optionnellement incrémental. Malheur procède d'abord en extrayant un petit sous-ensemble d'éléments qui représentent des noeuds typiques appelés *prototypes*. Ces prototypes sont extraits en $O(kn)$, où n est le nombre de maliciels à regrouper et k le nombre de prototypes. Cette technique permet de rapidement extraire un nombre relativement petit de prototypes dont chacun correspond à un groupe au début du regroupement, puis de classifier les éléments dans les groupes existants. Chaque groupe

est ensuite fusionné avec le plus proche groupe jusqu'à ce que la distance minimale entre deux groupes atteigne un seuil donné. Le temps d'exécution de cet algorithme est de $O(k^2 \log k + n)$, contrairement à la plupart des algorithmes de regroupement hiérarchique qui ont généralement une complexité de l'ordre de $O(n^2 \log n)$ [17]. En combinant le regroupement et la classification, Malheur est capable de traiter de grands ensembles de données [17].

2.1.4.2 Regroupement à travers le temps

La plupart des algorithmes de regroupement sont limités au nombre de données qu'une machine est capable de garder en mémoire, ce qui rend le regroupement de très larges ensembles de données infaisables. Certains algorithmes permettent l'évolution des groupes créés lors du regroupement initial.

Il existe des algorithmes de regroupement *incrémentaux*, c'est-à-dire un algorithme qui regroupe en plusieurs étapes (incréments) un ensemble de points. Cette technique est par exemple utilisée pour regrouper des grands ensembles de données qui ne peuvent pas être regroupés en une fois, mais aussi pour regrouper des données qui évoluent à travers le temps. Les algorithmes de regroupement incrémentaux font une première passe de regroupement régulier et par la suite permettent d'ajouter des points aux groupes existants après la création de ceux-ci [5]. Rieck *& al.* [17] proposent une approche incrémentale qui permet de reprendre un regroupement de maliciels déjà effectué pour y ajouter des nouveaux points et le faire évoluer. Ceci rend possible le regroupement de maliciels dans un système de production.

Cependant, les algorithmes de regroupement incrémentaux ne prennent pas en compte la possibilité de réassigner à un autre groupe un point déjà assigné. Cela veut dire que les points d'un groupe restent dans ce groupe, même si la distance entre un ou plusieurs points du groupe est plus grande que la distance entre ces derniers et un nouveau point. De plus, les algorithmes de regroupement incrémentaux ne permettent pas le retrait de points à travers le temps [6]. Pour parer à ces problèmes, il existe des algorithmes de regroupement *évolutif* qui permettent d'ajuster la position des points existants à travers le temps, tout en autorisant l'ajout et le retrait de points et ce en se basant sur l'historique du système [4]. Chi *& al.* [6] et Chakrabarti *& al.*

[4] proposent un modèle permettant l'évolution des groupes à travers le temps. Ils rendent possible le fait de réassigner des points d'un groupe à un autre en fonction de ses propriétés actuelles, mais aussi en fonction de son historique. Le modèle proposé prend en compte le fait que le nombre de points et le nombre de groupes changent à travers le temps et que ce changement n'est pas toujours régulier [6]. Chakrabarti *& al.* [4] posent la différence entre un algorithme de regroupement évolutif *connecté*, qui doit produire le regroupement à une étape t avant d'avoir accès aux données utilisées pour le regroupement à l'étape $t+1$ et un algorithme de regroupement évolutif *déconnecté* qui a accès à toutes les données avant de débuter le regroupement. Cependant, ces algorithmes de regroupement évolutifs n'ont pas été étudiés dans le domaine de la cyber-sécurité.

2.2 Extraction de signatures

En plus du regroupement, certains auteurs génèrent des signatures pour représenter les différents groupes. En effet, une fois les groupes formés, il est important de tirer de l'information de ces derniers. L'extraction de signatures, autrement dit l'extraction des actions d'un élément type pour un groupe, peut permettre d'automatiser la détection proactive des maliciels. Les signatures extraites peuvent être de plusieurs types : au niveau des actions sur l'hôte, au niveau du réseau ou même au niveau des bibliothèques importées.

Perdisci *& al.* [13] génèrent des signatures pour chaque groupe ou famille de maliciels généré et montrent que lors de l'étape d'extraction de signatures, il faut que les groupes soient compacts pour éviter les signatures trop génériques, qui pourraient ainsi générer trop de faux positifs. Pour procéder à l'extraction de signature, Perdisci *& al.* [13] extraient un élément représentatif de chaque groupe puis extraient la signature de ce dernier. Cette signature est ensuite passée à travers un filtre consistant de requêtes de traffic légitime et la signature est rejetée si des éléments du traffic légitime correspondent à la signature (faux-positifs).

FIRMA [16] est un outil qui permet de rassembler des échantillons de maliciels en groupes de familles et génère un ensemble de signatures pour chaque groupe. FIRMA [16] permet aussi d'extraire un ensemble de signatures associées à chaque groupe créé

par le regroupement. Les signatures générées sont aussi précises que celles générées manuellement par un analyste et leur format est compatible avec des systèmes de détection d'intrusions, mécanismes capables de détecter automatiquement des activités basées sur des règles ou des signatures.

La plupart des systèmes d'extraction de signatures actuels se concentrent sur les signatures réseau.

2.3 Étiquetage des données

Pour pouvoir vérifier la qualité du regroupement, il est important d'avoir accès à un ensemble d'entraînement ou à une façon de vérifier rapidement si le regroupement a produit des résultats valables. En effet, certains auteurs divisent leur ensemble de données en données d'entraînement, pour qui une étiquette est attribuée manuellement à chaque maliciel [10, 11, 18]. Cette façon d'opérer permet de vérifier la validité du modèle, mais requiert un étiquetage manuel, coûteux en temps. Malgré le fait que certaines initiatives tentent de normaliser le nommage des maliciels [2], chaque compagnie d'antivirus possède ses propres conventions de nommage. Sebastián *& al.* [19] proposent AVClass (voir section 1.2).

Conclusion

Dans cette section, nous avons pu remarquer que de plus en plus d'auteurs s'intéressent aux regroupements basés sur le comportement du maliciel pour permettre la détection des nouveaux maliciels. Les auteurs utilisent une approche en fonction de l'analyse statique, l'analyse dynamique ou sur une approche qui combine les deux types d'analyses. D'autre part, les algorithmes de regroupement utilisés diffèrent grandement et les résultats de ces derniers varient selon les données et les mesures de distance utilisées. Les algorithmes de regroupement évolutif étudiés dans ce chapitre ne sont pas utilisés spécifiquement pour le regroupement de maliciels et posent encore des problèmes quant à l'évolution des points par rapport aux groupes et à l'évolution

2. http://www.caro.org/articles/naming.html

de la taille des groupes. De surcroît, certains auteurs procèdent à l'extraction de signatures réseaux ou au niveau des actions du maliciel sur l'hôte à partir des groupes créés. Enfin, la plupart des auteurs considèrent l'utilisation d'ensembles d'entraînement manuellement étiquetés pour valider leur modèle.

Pour permettre un regroupement dans un système de production, il faut que de nombreux maliciels puissent être regroupés de façon rapide et évolutive en un nombre raisonnable de groupes. De plus, il faut que le regroupement de comportements similaires mais non égaux soit possible. Par exemple, la création d'un fichier dont le nom contient des éléments aléatoires devrait être regroupé avec d'autres maliciels utilisant la même stratégie, sans que l'aléatoire du nom de fichier soit pénalisant dans le calcul du regroupement. Enfin, il faut être capable d'extraire de l'information des groupes créés.

Notre approche, qui est étudiée plus en détail dans le prochain chapitre, s'intéresse à Malheur [17], l'algorithme utilisé ainsi que les variations et améliorations proposées.

Chapitre 3

Algorithme de regroupement

Dans notre revue de la littérature du domaine, nous avons pu remarquer que de nombreuses techniques existent pour regrouper des maliciels, en utilisant différents types d'analyses et différents algorithmes. Notre approche consiste à modifier Malheur [17], un algorithme hiérarchique qui permet de regrouper des maliciels de façon incrémentale. Dans ce chapitre, nous présentons l'algorithme Malheur [17] et les changements que nous y avons apportés.

Nous commençons tout d'abord par décrire les différentes façons de calculer la distance entre deux vecteurs de caractéristiques dans la section 3.1, puis nous présentons l'algorithme de regroupement dans la section 3.2, puis nous montrons notre approche de la visualisation des données dans la section 3.3.

3.1 Vecteurs de caractéristiques

Nous présentons ici trois façons de générer les vecteurs de caractéristiques et de calculer la distance entre ces derniers. La technique utilisée par l'algorithme Malheur [17] original est tout d'abord expliquée, puis suivie de deux variations.

Les fichiers donnés en entrée de l'algorithme de regroupement correspondent à une suite d'actions du maliciel délimitées par un symbole précis. En effet, pour regrouper des maliciels en fonction de leur comportement, un compte rendu des actions

du maliciel ou de ses caractéristiques est requis. Dans le cas d'un regroupement effectué grâce à des données provenant d'analyses statiques, le compte rendu prend par exemple la forme de chaînes de caractères ou d'en-têtes extraites du maliciel. Dans le cas de regroupement effectué avec des données provenant d'analyses dynamiques, le compte rendu ressemble par exemple à une liste d'actions effectuées par le maliciel ou encore à des informations reliées à l'état de la mémoire lors de l'exécution du maliciel. Dans ce mémoire, chaque ligne d'un fichier en entrée correspond soit à une caractéristique (bibliothèque importée ainsi qu'informations reliées à cette dernière), soit à une action du maliciel (appel réseau ou action au niveau de l'hôte) et est délimitée par un retour à la ligne (le format des fichiers utilisés est étudié plus en détail dans la section 4.1 et la section 4.2).

3.1.1 Algorithme original

Pour permettre de comparer les différentes façons de générer les vecteurs de caractéristiques ainsi que le calcul de leurs distances, nous avons inclus la technique que Malheur utilise pour générer les vecteurs de caractéristiques ainsi que pour calculer la distance entre ces derniers.

Calcul d'une caractéristique Pour calculer une caractéristique, l'algorithme Malheur effectue un hachage (MD5) de chaque n-gramme (voir sous-section 1.1.1.4). L'utilisation d'une fonction de hachage permet de réduire l'espace utilisé pour stocker la description du n-gramme ainsi que de réduire grandement le temps requis au calcul de la distance.

Vecteurs de caractéristiques Les vecteurs de caractéristiques sont des tableaux de deux dimensions contenant le MD5 de chaque n-gramme et la fréquence de ce dernier (voir tableau 3.1). Lors de la création du vecteur de caractéristiques, les caractéristiques sont calculées, puis les doublons présents dans ces dernières sont enlevés et la fréquence de chaque caractéristique pour un fichier est calculée, formant ainsi un tableau à deux dimensions représentant le vecteur de caractéristiques. Enfin, chaque caractéristique du vecteur est normalisée grâce à la normalisation L2 (voir sous-section 1.4.1.3).

tableau 3.1 – Représentation d'un vecteur de caractéristiques

n-gramme	fréquence du n-gramme
n-gramme	fréquence du n-gramme
n-gramme	fréquence du n-gramme
...	...

Distance entre deux vecteurs de caractéristiques Pour pouvoir calculer la distance entre deux vecteurs de caractéristiques, l'algorithme original utilise la distance euclidienne (voir section 1.4.1.3). Puisque la distance est calculée à partir des n-grammes du document, elle correspond donc à la distance entre deux fichiers complets.

Un désavantage de l'utilisation du hachage de la valeur tel que présenté dans sous-section 3.1.1 est que le moindre changement de la valeur peut résulter en un MD5 complètement différent. En effet, pour une phrase donnée, le changement d'un seul caractère change le MD5. L'utilisation d'une technique de recherche approximative permettrait de mitiger ce désavantage en effectuant un calcul de similitude entre deux caractéristiques.

3.1.2 Variations des vecteurs de caractéristiques

Grâce à l'utilisation de techniques de recherche approximative, il est possible que deux caractéristiques proches, mais pas exactement égales aient une distance faible, menant ainsi à un regroupement plus juste. Cette technique est utile lorsque deux caractéristiques diffèrent seulement légèrement, comme par exemple des dates ou nombres aléatoires qui changent à chaque exécution. La recherche approximative permettrait donc aux valeurs aléatoires ou changeantes d'avoir moins de poids sur le calcul de distance. Deux variations de l'algorithme ont été implémentées pour calculer la distance entre deux vecteurs de caractéristiques en utilisant cette technique.

3.1.2.1 Variation : Malheur SSDeep

Il existe de nombreux algorithmes de hachage flou. Parmi eux, nous avons choisi SSDeep [1], algorithme de hachage flou connu et pour lequel le calcul est assez rapide.

Calcul d'une caractéristique Le hachage flou effectue un hachage partie par partie du n-gramme, qu'on appelle une empreinte approximative. La caractéristique correspond donc à plusieurs empreintes, une pour chaque partie déterminée par l'algorithme à partir du n-gramme.

Vecteurs de caractéristiques Les vecteurs de caractéristiques sont des tableaux à une dimension contenant l'empreinte approximative de chaque n-gramme.

Distance entre deux vecteurs de caractéristiques Il est possible de calculer la dissimilitude entre deux caractéristiques en comparant chaque partie de l'empreinte approximative d'une caractéristique avec les parties de l'empreinte approximative de l'autre caractéristique. Une moyenne des dissimilitudes ainsi calculées est ensuite effectuée pour avoir la distance entre les deux vecteurs complets.

3.1.2.2 Variation : Malheur Dissimilitude cosinus

La fréquence des termes et la dissimilitude cosinus sont souvent utilisées dans la fouille de textes pour évaluer la similarité entre des séquences de textes. Puisque chaque n-gramme est une suite d'actions sous forme de texte, c'est aussi une suite de mots. L'utilisation de cette technique permet de regrouper des vecteurs de caractéristiques possédant des caractéristiques similaires mais non égales.

Calcul d'une caractéristique Chaque n-gramme est séparé en mots et la fréquence de chaque mot est calculée pour former un tableau où chaque ligne est composée du mot et de sa fréquence, comme illustré par le tableau 3.2.

Vecteurs de caractéristiques Les vecteurs de caractéristiques sont des tableaux pour lesquels chaque caractéristique est elle-même représentée par un tableau.

1. https://ssdeep-project.github.io/ssdeep/index.html

tableau 3.2 – Représentation d'une caractéristique utilisant la fréquence des termes

mot	fréquence du mot
mot	fréquence du mot
mot	fréquence du mot
...	...

Distance entre deux vecteurs de caractéristiques Entre les caractéristiques de deux vecteurs, la dissimilitude cosinus (voir section 1.4.1.3) est calculée. Une moyenne des dissimilitude cosinus est ensuite calculée pour avoir la distance entre les deux vecteurs complets.

Nous utilisons ces trois variations pour générer les vecteurs de caractéristiques et pour calculer la distance et ainsi déterminer si l'utilisation de recherche approximative permet d'obtenir un meilleur regroupement.

3.2 Algorithme

Nous utilisons dans notre approche une variation de l'algorithme Malheur [17]. Tel que décrit dans le chapitre 2, Malheur est un algorithme de regroupement hiérarchique ascendant (voir section 1.4 pour un rappel sur le regroupement). Malheur procède en choisissant un certain nombre d'éléments types appelés prototypes, éléments qui forment les groupes initiaux au lieu de former autant de groupes que d'éléments comme dans l'approche conventionnelle du regroupement hiérarchique, ce qui réduit le nombre de groupes initiaux. Les groupes les plus proches sont ensuite fusionnés successivement, comme dans un algorithme de regroupement hiérarchique habituel. Les éléments qui ne sont pas encore assignés à un groupe sont assignés au groupe le plus proche. La particularité de Malheur est de permettre un temps d'exécution de $O(k^2 \log k + n)$, où k et n sont respectivement le nombre de prototypes et le nombre d'éléments et ce, grâce à la sélection de prototypes qui permet de réduire le nombre de groupes initiaux.

L'algorithme utilise un fichier de configuration qui contient des valeurs comme par exemple la taille du n-gramme, le caractère qui délimite la fin d'une ligne, différents seuils, le type de liaison à effectuer, *etc.* L'algorithme procède en plusieurs étapes :

— **étape 1** – extraction des vecteurs de caractéristiques ;
— **étape 2** – extraction d'une liste de prototypes à partir de la liste de vecteurs de caractéristiques ;
— **étape 3** – création et fusion des groupes les plus proches à partir des prototypes ;
— **étape 4** – regroupement du reste des vecteurs et élimination des groupes ayant trop peu d'éléments.

Nous allons maintenant étudier chaque étape plus en détail.

Étape 1 : Extraction des vecteurs de caractéristiques

Algorithme 1 : Extraction des vecteurs de caractéristiques

Données : Fichier de configuration, fichiers contenant les données, n
Résultat : Ensemble des vecteurs de caractéristiques E

```
1  Initialisation de la configuration(fichier de configuration);
2  pour chaque f ∈ fichiers de données faire
3      lignes ← lire les lignes de f;
4      pour l ← 1 à nombre de lignes faire
5          n-gramme ← calculer le n-gramme de lignes[l] à lignes[l + n];
6          ajouter n-gramme à la liste n-grammes;
7      fin
8      ordonner alphabétiquement la liste n-grammes;
9      condenser la liste n-grammes;
10     normalisation L2 de la liste n-grammes ;
11     v ← la liste n-grammes;
12     ajouter v à E;
13 fin
14 retourner Ensemble des vecteurs de caractéristiques E
```

L'algorithme 1 représente les étapes suivies pour lire les fichiers en entrée et en extraire les vecteurs de caractéristiques. Pour chaque fichier, les n-grammes sont extraits du fichier, puis la liste de n-grammes est ordonnée de façon alphabétique. Ensuite, puisque la liste de n-grammes contient possiblement des duplicatas, cette dernière est condensée pour enlever les duplicatas et un compte de fréquences de n-grammes est mis en place. Par exemple, si un n-gramme A se trouve trois fois dans la liste de

n-grammes originale, seule une instance de A est conservée après condensation de la liste et la fréquence à laquelle A est présent dans la liste de n-grammes originale est gardée comme attribut du n-gramme. Par la suite, chaque n-gramme est normalisé grâce à la norme L2 pour simplifier et rendre plus rapides les calculs de distance effectués lors du regroupement. De plus, la normalisation est utile dans le cadre de l'application à des maliciels. En effet, si un maliciel effectue une suite d'actions de façon répétitive et si un second ne l'effectue qu'une seule fois, ces deux maliciels sont considérés comme similaires puisqu'ils ont un comportement en commun. La liste de n-gramme résultante forme le vecteur de caractéristiques.

Calcul des *n*-grammes Les n-grammes sont calculés à partir des fichiers utilisés en entrée de l'algorithme et le n étant fourni dans un fichier de configuration.

Pour calculer les n-grammes associés à cette séquence d'instructions, une fenêtre coulissante de taille n est utilisée. Cette fenêtre coulissante commence par les n premières lignes du fichier, ce qui équivaut au premier n-gramme, puis avance d'une ligne, ce qui correspond au deuxième n-gramme et ainsi de suite jusqu'à la dernière ligne. La figure 3.1 illustre cette technique dans laquelle chaque rectangle de couleur représente un bi-gramme ($n = 2$), allant du premier (en rouge) jusqu'au dernier (en vert).

Contrairement à l'approche qui consiste à générer toutes les combinaisons possibles de lignes, l'utilisation d'une telle fenêtre coulissante permet de marquer l'importance de l'ordre entre les lignes, ainsi que de réduire le temps d'exécution. En effet, dans un fichier où l'ordre des lignes a une importance, extraire uniquement les n-grammes qui respectent cet ordre et non tous les n-grammes revient à procéder à une présélection et à une approximation de l'ensemble des caractéristiques qui forment le vecteur de caractéristiques.

Pour calculer un n-gramme, l'algorithme Malheur sélectionne n lignes d'un fichier et les concatène pour former un n-gramme. Chaque n-gramme forme une des caractéristiques d'un vecteur de caractéristiques grâce à une des trois techniques présentées dans la section 3.1. Une fois que l'ensemble des vecteurs de caractéristiques sont formés, le regroupement peut commencer.

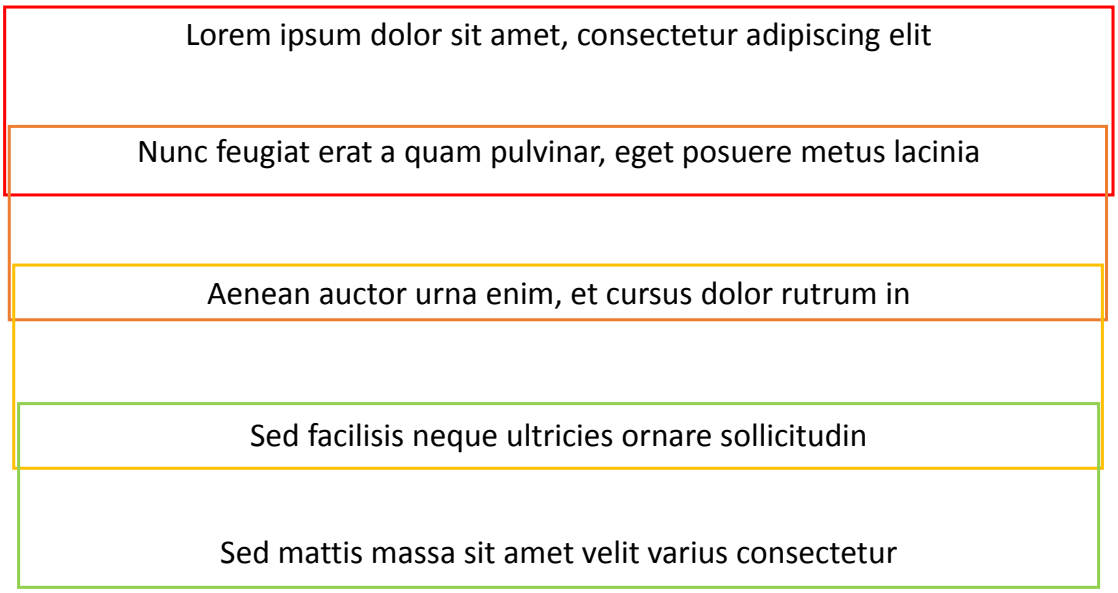


figure 3.1 – Représentation d'une fenêtre coulissante de taille $n = 2$

Étape 2 : Extraction des prototypes

Un certain nombre d'éléments types appelés prototypes sont tout d'abord extraits de l'ensemble des vecteurs de caractéristiques. Ces prototypes correspondent aux éléments les plus éloignés les uns des autres et chaque prototype représente le centre initial d'un groupe. Le premier élément est choisi au hasard et ajouté à la liste de prototypes. Ensuite, la distance de chaque vecteur de caractéristiques par rapport à ce prototype est calculée et le vecteur v de caractéristiques ayant la distance la plus élevée par rapport au prototype est choisi et ajouté à la liste de prototypes. Ce vecteur v devient alors l'élément auquel tous les vecteurs non-prototypes restants sont comparés. Cette technique permet de réduire le nombre d'éléments à comparer entre eux, tout en s'assurant de choisir l'élément le plus loin du dernier prototype comme prochain prototype. Comparer les éléments au dernier prototype choisi seulement au lieu de les comparer à tous les prototypes permet d'approximer le calcul de l'élément non-prototype le plus lointain des autres. Ceci est répété jusqu'à ce que la distance entre le dernier prototype choisi, c'est-à-dire l'élément le plus éloigné des autres vecteurs de caractéristiques, soit en dessous d'un seuil de distance donné. L'algorithme

Algorithme 2 : Extraction des prototypes

Données : Ensemble des vecteurs de caractéristiques E
Résultat : Ensemble des prototypes P

```
1  v ← élément au hasard parmi E;
2  tant que v.distance > seuil faire
3  |  ajouter v à P;
4  |  retirer v de E;
5  |  pour chaque vecteur ∈ E faire
6  |  |  vecteur.distance ← distance(vecteur, v);
7  |  fin
8  |  v ← vecteur ayant la distance maximum parmi E;
9  fin
10 retourner Ensemble des prototypes P
```

2 représente la marche à suivre pour extraire les prototypes.

Étape 3 : Création et fusion des groupes

L'algorithme utilisé étant ascendant, chaque prototype forme un groupe à lui seul au début de la création des groupes et peut être vu comme le centre du groupe. Une matrice de distances entre chaque prototype est calculée pour permettre de fusionner ensemble les prototypes les plus proches les uns des autres. Les deux groupes pour lesquels leur prototype sont les plus proches sont fusionnés ensemble. Lors de cette étape, un des deux prototypes (p_A) est gardé comme prototype du groupe tandis que l'autre (p_B) devient un élément régulier du groupe et n'est plus considéré comme un prototype. Selon le mode de liaison (voir sous-section 1.4.1.1), la distance minimale, moyenne ou maximale entre p_A et p_B est recalculée et la matrice des distances est mise à jour pour le prototype conservé p_A. Ces étapes sont répétées jusqu'à ce que la distance minimum entre les prototypes soit supérieure à un seuil donné. L'algorithme 3 représente l'étape de création et de fusion des groupes.

Étape 4 : Regroupement du reste des vecteurs

Ayant une liste de groupes constitués d'un ou plusieurs prototypes, il ne reste plus qu'à ajouter le reste des vecteurs qui ne sont pas dans un groupe aux groupes

Algorithme 3 : Création et fusion des groupes

Données : Ensemble des prototypes P, ensemble des vecteurs de caractéristiques E
Résultat : Ensemble des groupes G

$G \leftarrow$ initialiser chaque groupe avec un prototype;
distance_matrice $\leftarrow$ calculer les distances entre les prototypes;
tant que *distance minimale de distance_matrice* $<$ *seuil* **faire**
    p_A, $p_B \leftarrow$ prototypes entre lesquels la distance de *distance_matrice* est minimale;
    enlever le groupe de p_B de G;
    enlever p_B de P;
    ajouter p_B au groupe de p_A;
    $distance\_matrice_{p_A} \leftarrow$ calculer les distances entre p_A et p_B selon le mode de liaison;
    retirer p_B de *distance_matrice*
fin
retourner Ensemble des groupes G

Algorithme 4 : Ajout du reste des vecteurs de caractéristiques aux groupes

Données : Ensemble des groupes G, ensemble des vecteurs de caractéristiques E
Résultat : Ensemble des groupes G, ensemble des vecteurs de caractéristiques rejetés R

pour chaque $v \in E$ **faire**
    groupe_le_plus_proche $\leftarrow$ trouver le plus proche prototype pour v;
    si *distance(groupe_le_plus_proche, v)* $<$ *seuil* **alors**
        ajouter v au groupe le plus proche;
    sinon
        ajouter v à R;
    fin
fin
ajouter à R les éléments des groupes dont le nombre d'éléments est en dessous d'un seuil donné;
retourner Ensemble des groupes G, Ensemble des éléments rejetés R

existants. Pour ce faire, la distance entre chaque vecteur non-prototype et tous les prototypes est calculée. Le vecteur est alors assigné au groupe pour lequel sa distance avec le prototype est la plus faible, sauf si cette distance la plus faible est supérieure à un seuil donné, auquel cas le vecteur n'est pas assigné à un groupe et est mis de côté, gardé pour des regroupements ultérieurs. Enfin, quand tous les vecteurs ont été assignés à un groupe, les groupes ayant un nombre d'éléments inférieur à un certain seuil sont rejetés. L'algorithme 4 reflète l'étape de regroupement du reste des vecteurs.

À la fin de cette étape, le regroupement est complété et les vecteurs et leur assignation sont écrits dans un fichier pour pouvoir les relire en mémoire et continuer le regroupement dans le cas d'un regroupement incrémental.

3.3 Visualisation

Pour rendre le résultat du regroupement visualisable par un humain, nous transformons les résultats du regroupement en graphe Gephi [2]. Tous les éléments sont convertis au format *gexf* qui est lisible par Gephi et chaque noeud contient certaines informations, comme par exemple son étiquette, le prototype qui lui est associé, *etc.*

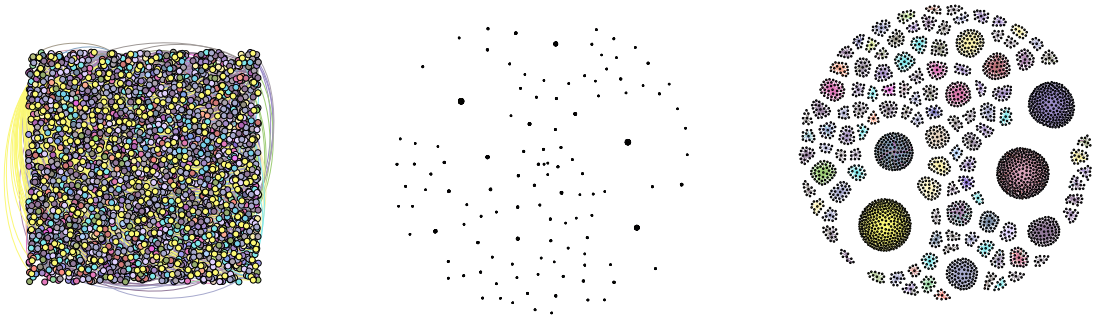
Une fois que tous les tuples sont convertis en noeuds, le graphe est exporté au format *gexf*. Ensuite, deux algorithmes fournis avec Gephi sont appliqués au graphe pour organiser les noeuds en groupes sous une forme visuellement interprétable :

— ForceAtlas2 [3], qui permet d'éloigner les points et de séparer les groupes. Cet algorithme est appliqué pendant 30 secondes, le temps nécessaire pour séparer de façon raisonnable les groupes et les points ;

— FruchtermanReingold [4], qui permet de rassembler les points en groupes et de les organiser pour que les groupes soient visibles à l'oeil.

La figure 3.2 illustre une succession de graphes aux différentes étapes du processus de visualisation. La figure 3.2a présente le graphe après la conversion au format *gexf*, c'est-à-dire sans qu'aucune organisation des points ne soit faite. Tous les éléments des groupes sont placés à un endroit par défaut et il est difficile de visualiser les

2. Logiciel utilisé pour visualiser les graphes, https://gephi.org/
3. https://github.com/gephi/gephi/wiki/Force-Atlas-2
4. https://github.com/gephi/gephi/wiki/Fruchterman-Reingold



(a) Graphe avant d'appliquer les algorithmes

(b) Graphe après ForceAtlas2

(c) Graphe après ForceAtlas2 suivi de Fruchterman-Reingold

figure 3.2 – Algorithmes appliqués aux graphes pour les organiser visuellement en groupes

différents groupes. La figure 3.2b montre le graphe après application de l'algorithme ForceAtlas2, qui sépare les groupes les uns des autres. Enfin, l'effet de l'algorithme FruchtermanReingold est illustré dans la figure 3.2c. Les groupes sont alors organisés de telle sorte que chaque élément des groupes soit visible à l'oeil, rendant l'observation par un humain possible.

Il est possible d'afficher les étiquettes des points comme illustré dans la figure 3.3, rendant ainsi l'étude des éléments des groupes et la détection des éléments qui ne devraient pas être dans un groupe plus rapide. De plus, la couleur d'une famille reste constante d'un graphe à l'autre, permettant de reconnaître plus facilement les groupes d'une exécution à l'autre.

Conclusion

Dans cette section, nous avons étudié en détail le fonctionnement de l'algorithme Malheur et de ses particularités. Cet algorithme permet un temps d'exécution plus faible que les algorithmes habituellement utilisés en procédant à une sélection des

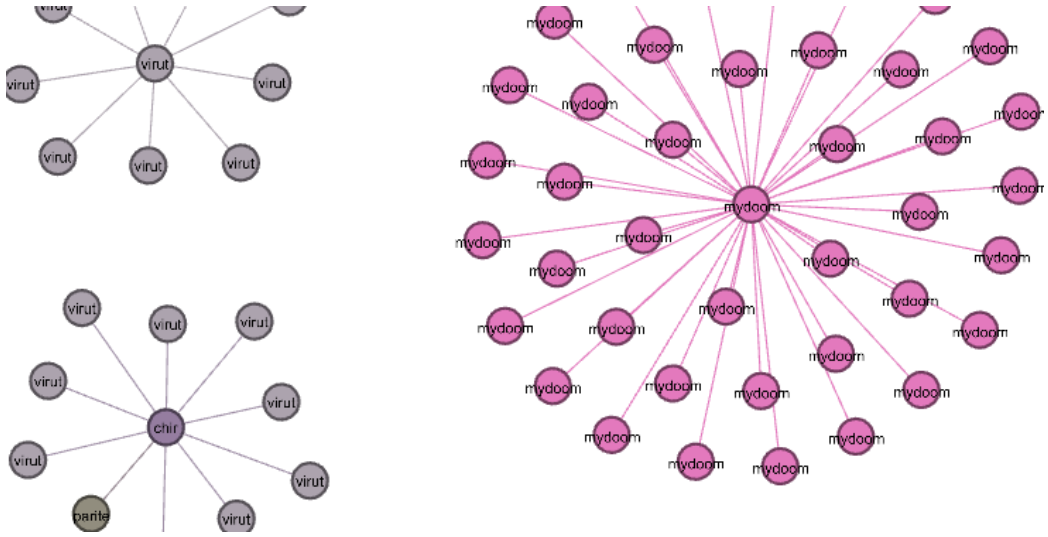


figure 3.3 – Partie d'un graphe avec les étiquettes affichées

n-grammes extraits. D'autre part, nous avons décrit les modifications apportées à cet algorithme. En effet, l'utilisation d'une fonction de hachage lors du calcul des n-grammes ne permet pas de variation dans le n-gramme lui-même. L'utilisation de recherche approximative permet donc d'évaluer la similarité entre deux lignes de textes, permettant de regrouper les textes similaires en plus des textes égaux. Enfin, nous avons abordé la création d'un graphe grâce au logiciel Gephi permettant la visualisation du regroupement. En effet, le résultat de l'algorithme est un ensemble de noeuds organisés en groupes et exporté vers un fichier texte. L'utilisation d'un graphe pour visualiser les groupes ainsi créés permet de déterminer rapidement si les groupes formés sont homogènes et d'explorer les noeuds mal placés par le regroupement.

L'application de cet algorithme au domaine de la sécurité informatique et des maliciels est étudiée en détail dans le prochain chapitre.

Chapitre 4

Présentation de notre solution : CASTOR

L'algorithme utilisé dans ce mémoire est entre autres applicable au domaine de la sécurité informatique. En effet, un avantage important de Malheur est de pouvoir permettre un temps d'exécution de $O(k^2 \log k + n)$, où k et n sont respectivement le nombre de prototypes et le nombre d'éléments. Le nombre de maliciels étant grand, un algorithme s'exécutant en un temps raisonnable est crucial pour le regroupement de ces derniers.

Nous commençons tout d'abord par décrire les données utilisées dans la section 4.1, puis abordons le traitement de ces dernières pour les formater avant d'y appliquer l'algorithme dans la section 4.2. Enfin, nous décrivons notre implémentation nommée *CASTOR* dans la section 4.3.

4.1 Jeux de données

La base de données du Centre canadien de réponse aux incidents cybernétiques (CCRIC) contient des informations et des résultats d'analyse de millions de maliciels. En revanche, travailler et extraire un modèle et des résultats d'une telle masse de données est difficile. C'est pourquoi nous avons choisi de travailler avec un sous-ensemble des données.

Pour chaque maliciel, trois résultats d'analyses sont extraits de la base de données :

— le résultat de l'enregistrement des actions sur l'hôte du maliciel ;
— le résultat de l'étude de son en-tête de fichier *Portable Executable (PE)* ;
— le résultat de la surveillance réseau.

Ces trois comportements nous permettent d'étudier un maliciel sous plusieurs angles, ce qui permet de mieux cerner le comportement du maliciel. Par exemple, un rançongiciel (voir sous-section 1.1.1) présente certains imports comme des bibliothèques d'encryption, effectue des communications réseaux pour rejoindre son serveur de commande et de contrôle et fait la liste des fichiers présents sur la machine qu'il tente d'infecter. Ensemble, ces comportements permettent de classifier le maliciel de façon plus précise.

4.1.1 Aspect hôte

Le regroupement peut être fait en fonction du comportement que le maliciel a sur la machine ; ce comportement est représenté sous forme d'actions de la part du processus malicieux sur un autre processus, un fichier, un dossier, *etc.* (voir section 1.1.1.1). Par exemple, un maliciel peut lister tous les fichiers présents sur un ordinateur, tandis que d'autres ajoutent des clés dans le registre ou encore démarrent un processus enfant.

Le tableau 4.1 montre l'exemple d'un maliciel qui crée, modifie et exécute des fichiers, qui appelle une interface de programmation (*API*) et qui cherche à trouver la valeur d'une clé de registre. Dans la séquence d'actions présentée dans le tableau 4.1, il est important de remarquer l'ordre. En effet, si deux actions sont toujours effectuées dans le même ordre, elles peuvent être reliées. De plus, le processus créé est important et le nom et l'endroit du fichier créé peuvent être utilisés comme indicateurs pour reconnaître ce type de maliciel. Dans cet exemple, le maliciel va chercher le nom de l'ordinateur (ligne 1), puis crée un fichier (ligne 2), pour ensuite démarrer ce dernier (ligne 3). Le processus ainsi lancé supprime un fichier nommé «file.txt» (ligne 4) et arrête le processus nommé «process.exe».

tableau 4.1 – Exemple du comportement local d'un maliciel

	Processus	Action	Valeur
2	malware.exe	ReadReg	`\REGISTRY\MACHINE\SYSTEM\ ControlSet001\Control\ComputerName\ ActiveComputerName\"ComputerName"`
3	malware.exe	CreateFile	`C:\Windows\lsass.exe`
4	maware.exe	StartProcess	`C:\Windows\lsass.exe`
5	lsass.exe	DeleteFile	`C:\Windows\file.txt`
6	lsass.exe	StopProcess	`C:\Windows\process.exe`
7	...	...	...

4.1.2 Aspect en-tête

Il est possible de regrouper les maliciels en fonction de leur en-tête des fichiers *Portable Executable (PE)*. Étudier les en-têtes des fichiers *PE* peut apporter de l'information à propos du maliciel et de la façon dont il a été créé. Par exemple, dépendamment de la façon dont sont écrits les imports dans l'en-tête, il est possible de reconnaître le style de code utilisé par le programmeur du maliciel. Il est aussi possible de noter le compilateur utilisé pour compiler le maliciel, puisque chaque compilateur laisse une trace spécifique et a différentes options qui sont visibles dans l'en-tête du fichier. De plus, la plupart des programmeurs ont tendance à réutiliser des bibliothèques qu'ils ont utilisées dans du code précédent, ce qui rend possible le regroupement à partir des imports de bibliothèques.

Le tableau 4.2 donne un exemple des bibliothèques importées par un maliciel de la famille «*cerber*». Ce maliciel utilise la bibliothèque «*ADVAPI32.dll*» pour appeler des méthodes comme par exemple «*RegOpenKeyExW*», «*RegCloseKey*», «*RegCreateKeyExW*». Ces fonctions montrent de l'information à propos des actions effectuées par le maliciel. Dans ce cas-ci, le maliciel tente de modifier le registre. Tous ces imports et ces caractéristiques forment une signature spécifique qui est un aspect intéressant lors de regroupement.

4.1.3 Aspect réseau

Finalement, il est aussi possible de regrouper les maliciels en fonction du traffic réseau qu'ils génèrent. La plupart des maliciels essayent de se connecter à un site

tableau 4.2 – Exemple de bibliothèques importées dans une en-tête de fichier *PE*

1	ADVAPI32.dll
2	COMCTL32.dll
3	KERNEL32.dll
4	USER32.dll

internet ou à un serveur pour pouvoir envoyer ou télécharger des données. Si deux maliciels se connectent à la même adresse IP, ces derniers peuvent être liés d'une façon ou d'une autre, par exemple en téléchargeant un fichier du même site ou utilisant le même serveur de commande et de contrôle (voir sous-section 1.1.1.1).

Le tableau 4.3 montre l'extrait de traces réseaux d'un maliciel. Les adresses IP «*INCONNU*» montrent qu'un domaine n'a pas pu être résolu et «*VIDE*» est écrit pour montrer qu'aucune donnée n'a été envoyée dans la requête. Grâce à cet extrait de traces réseaux, il est possible de remarquer que les deux premiers appels ne devraient pas être pris en compte par l'algorithme de regroupement puisque ces derniers sont des appels à des sites ou adresses IP connus. En revanche, les appels suivants ne sont pas communs et peuvent mener à d'autres maliciels similaires.

tableau 4.3 – Exemple de capture de traces réseaux

	IP	**Domaine**	**Protocol**	**Port**	**Données**
2	8.8.8.8	8.8.8.8	UDP	53	VIDE
3	184.150.152.183	google.com	TCP	80	GET / HTTP/1.1 ~~Host : google.com ~~User-Agent :
4	INCONNU	cpswyq.com	TCP	443	VIDE
5	INCONNU	eiuxag.com	TCP	443	VIDE
6	INCONNU	ejztdo.com	TCP	443	VIDE
7	...	...	...	...	...

Pour conclure, nous avons retenu l'aspect en-tête, l'aspect réseau et l'aspect hôte comme données utilisées dans l'algorithme de regroupement. En effet, ces trois aspects provenant de l'analyse statique et de l'analyse dynamique permettent d'évaluer le comportement du maliciel selon des angles distincts et assez génériques pour que

ces actions soient présentes dans la majorité des maliciels, rendant possible le regroupement de nombreux maliciels.

4.2 Traitement des données

Nous utilisons un jeu de données (voir section 4.1) provenant du CCRIC qui contient les résultats d'analyses sous forme de rapports au format XML ou JSON. Chaque résultat d'analyse possède son propre formatage, rendant nécessaire d'effectuer un traitement de ces résultats avant de les utiliser comme entrée de l'algorithme de regroupement.

4.2.1 Formatage des fichiers en entrée

Pour en extraire une partie que nous utilisons dans notre regroupement et pour formater les rapport d'une façon uniforme, nous effectuons un traitement sur les résultats d'analyses. Pour chaque aspect des données (en-tête, réseau et hôte, voir respectivement sous-section 4.1.2, sous-section 4.1.3 et sous-section 4.1.1), un fichier CSV est généré à partir du rapport contenant des informations à propos du processus source, de l'action effectuée et de la valeur de l'argument.

4.2.1.1 Aspect hôte

Le tableau 4.4 [1] définit les différentes valeurs possibles selon le type d'évènement possible pour un rapport d'actions provenant de l'hôte. Le nom du processus source correspond au processus qui génère l'évènement, c'est-à-dire le maliciel ou un processus démarré par ce dernier. Le mode est l'action effectuée, comme par exemple «*find*», «*open*» ou «*close*» pour un fichier, «*queryvalue*», «*deleteval*» ou «*setval*» pour une clé de registre, *etc.* La colonne nommée «Valeur» correspond à la valeur de la cible de l'évènement, comme par exemple le chemin du fichier à créer dans le cas d'une création de fichier, le chemin de la clé dont la valeur est changée, *etc.* Le tableau 4.5

1. À noter que la colonne «Type d'évènement» n'est pas présente dans les fichiers CSV, elle est ajoutée dans ce tableau pour faciliter la compréhension du lecteur.

montre un exemple d'évènements et leurs valeurs telles qu'elles sont écrites dans le fichier CSV généré.

tableau 4.4 – Format du rapport de données provenant de l'hôte

Type d'évènement	Processus source	Mode	Valeur
Fichier	<nom du processus source>	folder_<mode>	<chemin jusqu'au fichier cible>
Registre	<nom du processus source>	regkey_<mode>	<chemin jusqu'à la clé>
API	<nom du processus source>	apicall_<nom de l'API>	<nom du fichier dll> - <nom de l'API>
Processus	<nom du processus source>	process_<mode>	<nom du processus cible>
Réseau	<nom du processus source>	network_<mode>	

tableau 4.5 – Exemple de rapport CSV de données provenant de l'hôte

Type d'évènement	Processus source	Mode	Valeur
API	SUSPICIOUS.exe	apicall_GetSystemTime	kernel32.dll - GetSystemTime
Fichier	SUSPICIOUS.exe	folder_open	C:\DOCUME~1\admin\LOCALS~1\Temp\ACTIVEDS.dll
Réseau	SUSPICIOUS.exe	network_http_request	
Processus	SUSPICIOUS.exe	process_started	C:\DocumentsandSettings\admin\ApplicationData\52A198F4-9\bootok.exe
Registre	bootok.exe	regkey_setval	\REGISTRY\USER\S-1-5-21-143\Software\Microsoft\Windows\CurrentVersion\InternetSettings\"ProxyServer"=10.0.0.2:8080

4.2.1.2 Aspect en-tête

De même, le tableau 4.6[1] montre le formatage utilisé pour transformer les données provenant d'analyses de bibliothèques et imports en un fichier CSV. Le nom de la bibliothèque dans la colonne «Bibliothèque importée» correspond aux noms des bibliothèques importées, tandis que la colonne nommée «Méthode utilisée» dénote les méthodes importées par ces dernières et la colonne «Point d'entrée» indique l'adresse mémoire de leur point d'entrée. Le tableau 4.7 est un exemple d'évènements présent dans un des fichiers CSV générés.

tableau 4.6 – Format du rapport de données provenant des bibliothèques et imports

Type d'évènement	Bibliothèque importée	Méthode utilisée	Point d'entrée
Bibliothèque	<nom de la bibliothèque>	<nom de la méthode>	<adresse du point d'entrée>

tableau 4.7 – Exemple de rapport CSV de données provenant des bibliothèques et imports

Type d'évènement	Bibliothèque importée	Méthode utilisée	Point d'entrée
Bibliothèque	MSVBVM60.DLL	__vbaAryMove	0x401018

4.2.1.3 Aspect réseau

Le tableau 4.8[1] illustre le formatage utilisé dans les fichiers CSV contenant les données provenant d'analyses d'appels réseaux. La colonne nommée «Adresse réseau» contient le domaine ou l'adresse IP que le maliciel a tenté d'atteindre. Si l'appel réseau a été effectué vers un domaine, alors le domaine est présent, sinon l'adresse IP que le maliciel a tenté d'atteindre directement est indiquée dans cette colonne. La colonne «Protocole/port» correspond à une concaténation du protocole utilisé ainsi que le port auquel l'appel réseau a été fait. Enfin, la colonne «Données» dénote les données envoyées avec la requête réseau. Dans cette colonne, « *VIDE*» est écrit pour montrer qu'aucune donnée n'a été envoyée dans la requête. Le tableau 4.9 est un exemple d'évènements générés par un maliciel formaté selon le format défini.

tableau 4.8 – Format du rapport de données provenant des appels réseau

Type d'évènement	Adresse réseau	Protocole/port	Données
Appel réseau	<domaine ou adresse IP>	<protocole>/<port>	<données>

tableau 4.9 – Exemple de rapport CSV de données provenant des appels réseau

Type d'évènement	Adresse réseau	Protocole/port	Données
Appel réseau	72.137.27.17	tcp/139	VIDE
Appel réseau	195.27.190.10	tcp/139	VIDE
Appel réseau	109.146.113.54	udp/137	VIDE
Appel réseau	downloads.airdwnlas.com	http/80	GET /get/ ?key=de84ece306d8c887c627bc40ba787a85e HTTP/1.1 Host : downloads.airdwnlas.com User-Agent : Launcher Get Log Level

4.2.2 Uniformisation des données

L'étape d'uniformisation permet au regroupement de ne pas tenir compte des valeurs spécifiques ou aléatoires qui changent de maliciel en maliciel ou d'exécution en exécution, biaisant ainsi le regroupement. Par exemple, si un maliciel nomme un fichier avec la date du moment de l'exécution alors le fichié créé aura un nom différent

à chaque exécution, ce qui en fait un attribut peu approprié au regroupement. Une façon d'éviter ces valeurs est de remplacer la date dans le nom du fichier par une valeur invariante, comme par exemple «DATE».

Dans le jeu de données du CCRIC, chaque maliciel est désigné par son identifiant unique, soit son MD5, valeur propre et unique à chaque maliciel. Deux maliciels avec des MD5 différents peuvent appartenir à la même famille (voir 1.1.2.2). Ainsi, nous effectuons comme uniformisation le remplacement du MD5 du maliciel par *«SUSPICIOUS.exe»*. Cette uniformisation permet au regroupement de traiter le nom du maliciel comme égal pour chaque maliciel et ne pas être influencé par le nom original du maliciel, c'est-à-dire le MD5 ou encore l'identifiant unique du maliciel.

Une fois les données uniformisées et formatées, les fichiers CSV générés sont utilisés comme entrée de l'algorithme de regroupement.

4.2.3 Étiquetage des données

Pour nous permettre de vérifier le résultat du regroupement obtenu, nous étiquetons les maliciels avec le nom de la famille auquel ce dernier appartient. Comme mentionné dans la section 1.2, l'étiquetage des données est basé sur les résultats d'antivirus et chaque antivirus possède ses propres conventions de nommage, ce qui rend la détermination d'un nom commun parmi tous les résultats difficile. Nous procédons à l'étiquetage des maliciels avec un nom le plus probable grâce à une variation du programme AVClass [19] (voir section 1.2).

Tout d'abord, nous utilisons une liste de variantes tirée de la base de données du CCRIC, en plus d'une liste de parties génériques mise à jour (e.g., ajout de «*fileinfector*» à cette liste). AVClass possède un seuil d'acceptation d'un nom inconnu en dessous duquel un nom n'est pas choisi lors de l'étape du vote (voir figure 1.2, étape f). Par défaut, ce seuil est de 1. Nous avons modifié AVClass pour être capable de changer la valeur du seuil d'acceptation d'un nom grâce à un fichier de configuration.

De plus, nous avons changé la façon dont AVClass traite les noms lors d'une égalité. Originalement, le programme ordonne les votes (voir figure 1.2, étape f) par nombre de votes décroissant. Dans le cas où deux noms possèdent le même nombre de votes, le nom est alors choisi par ordre alphabétique décroissant. Pour obtenir des

meilleurs résultats en cas d'égalité, nous avons modifié AVClass pour que :

- si un des deux noms en égalité est dans la liste de variantes, nous choisissons celui-ci ;
- si les deux noms en égalité sont dans la liste de variantes, nous vérifions les données présentes dans une table de la base de données qui représente les relations entre les maliciels (tel qu'énoncé dans la sous-section 1.1.1.3, certains maliciels en téléchargent d'autres et cette relation permet de décider quel nom choisir si une relation est connue entre les deux noms en égalité) ;
- si aucun des noms n'est dans la liste de variantes, nous procédons comme originalement, c'est-à-dire en choisissant par ordre alphabétique décroissant.

Les fichiers CSV générés et uniformisés forment donc l'entrée du regroupement, tandis que les étiquettes permettent de vérifier le résultat du regroupement et de calculer les mesures de qualité de ce dernier.

Dans cette section, nous avons décrit le jeu de données utilisé pour le regroupement. Ces données correspondent aux résultats de trois types d'analyses différentes :

- au niveau des actions sur l'hôte, c'est-à-dire des actions que le maliciel a effectué sur l'hôte lorsqu'il a été exécuté dans un bac à sable (analyse dynamique) ;
- au niveau des en-têtes, c'est-à-dire des bibliothèques importées par le maliciel pour pouvoir s'exécuter (analyse statique) ;
- au niveau des actions réseaux, c'est-à-dire des actions réseaux que le maliciel a effectué lorsqu'il a été exécuté dans un bac à sable (analyse dynamique).

Nous avons aussi présenté la façon d'étiqueter les maliciels, utilisant une variation du programme AVClass [19] (voir section 1.2). Trouver un ensemble de données de tests étiquetées assez grand et récent est souvent difficile à trouver. AVClass permet de déterminer l'étiquette d'un maliciel la plus probable et ainsi d'obtenir facilement un ensemble de tests pour lequel les étiquettes sont probablement bonnes.

4.3 Notre implémentation : *CASTOR*

Le programme qui a été développé se nomme ***C****lustering* ***A****lgorithm for* ***S****ecurity* ***T****hreats* ***O****rganization and* ***R****eporting* (CASTOR). Le but du programme est d'effec-

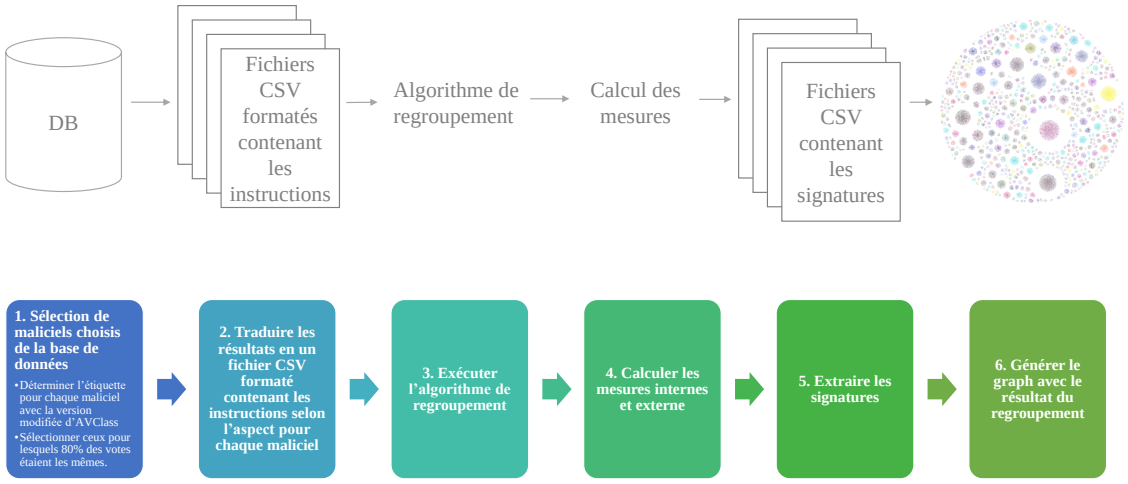


figure 4.1 – Diagramme représentant l'exécution du programme CASTOR pour un aspect

tuer les étapes de sélection des maliciels à regrouper, le traitement des données, le regroupement, l'extraction de signatures et la génération de graphe automatiquement.

Comme représenté sur la figure 4.1, CASTOR exécute les étapes suivantes pour chaque aspect (hôte, en-tête et réseau) :

1. sélectionner et extraire les résultats d'analyses pour les échantillons choisis ;
2. traduire ces résultats d'analyses en un fichier CSV pour chaque maliciel ;
3. exécuter l'algorithme de regroupement en donnant le dossier contenant les fichiers CSV en entrée ;
4. calculer des mesures internes et externes choisies pour vérifier la qualité du regroupement ;
5. extraire les signatures des groupes ;
6. convertir le résultat de l'algorithme de regroupement vers un graphe.

Voici la description des étapes du programme avec plus de détails.

Étape 1 – sélection et extraction des résultats d'analyses pour les maliciels choisis

Tout d'abord, une liste de maliciels est extraite de la base de données. Les maliciels choisis sont ceux pour lesquels plus de 80% des antivirus s'accordaient sur le nom du maliciel lorsqu'AVClass a été appliqué lors de l'étiquetage (voir sous-section 4.2.3). Cette sélection permet de travailler avec un ensemble de maliciels pour lesquels l'étiquette est très probablement bonne, ce qui est important lors de la vérification de la qualité du regroupement.

Étape 2 – traduction des résultats d'analyses dans un fichier CSV pour chaque maliciel

Pour chaque maliciel, un fichier CSV contenant les représentations issues d'analyses est créé et formaté. Ce fichier contient donc les commandes exécutées ou les appels systèmes effectués par le maliciel ainsi que les valeurs passées en arguments selon l'aspect choisi (voir section 4.2). Lors de cette étape, il est possible d'activer ou de désactiver l'uniformisation des données (voir sous-section 4.2.2), pour permettre de déterminer l'influence de celle-ci sur le regroupement. Cette uniformisation consiste à changer le nom du maliciel, originalement représenté par son empreinte unique (MD5) pour un nom standard, ainsi que standardiser les dates et les valeurs aléatoires.

Ces fichiers CSV sont organisés en dossiers par étiquettes. Par exemple, un maliciel du rançongiciel *Cerber* est placé dans le dossier *Cerber*, à côté de tous les autres fichiers CSV ayant la même étiquette.

Étape 3 – exécution de l'algorithme de regroupement en donnant le dossier racine qui contient les fichier CSV

L'algorithme (voir section 3.2) est exécuté avec le dossier racine qui contient les fichiers CSV, c'est-à-dire le dossier qui contient les dossiers d'étiquettes, ces derniers contenant eux-mêmes les fichiers CSV. À cette étape, il est possible de choisir d'utiliser l'algorithme Malheur original, Malheur SSDeep ou encore Malheur dissimilitude cosinus (voir section 3.1).

Étape 4 – calcul des mesures internes et externes choisies pour vérifier la qualité du regroupement

Pour permettre la comparaison de deux regroupements et pour en évaluer la qualité, nous utilisons plusieurs mesures (voir sous-section 1.4.2) : une matrice de confusion et la précision comme mesures externes (voir sous-section 1.4.2.1) et l'indice Silhouette comme mesure interne (voir sous-section 1.4.2.2).

Pour créer la matrice de confusion et calculer la précision, nous utilisons les étiquettes préalablement créées pour chaque maliciel (voir sous-section 4.2.3).

L'indice Silhouette mesure la cohésion intragroupe et l'absence de cohésion intergroupe, ce qui veut dire que plus les éléments d'un groupe se ressemblent et plus les groupes sont différents entre eux, plus l'indice Silhouette est élevé.

Étape 5 – extraction des signatures

Pour extraire les différentes signatures des groupes, les fichiers utilisés comme entrée du regroupement sont lus groupe par groupe. Il est ainsi possible de calculer la fréquence de chaque instruction dans le groupe. Les instructions revenant plus de 75% du temps sont alors choisies comme signature du groupe et exportées vers un fichier CSV. Les signature extraites doivent idéalement représenter le groupe et être assez spécifiques pour ne pas représenter d'autres groupes.

Étape 6 – conversion du résultat de l'algorithme en un graphe

Nous utilisons Gephi [2] pour visualiser les résultats de l'algorithme (voir section 3.3).

Tel que montré sur la figure 4.2, le graphe résultant d'un regroupement avec la configuration par défaut de Malheur montre plusieurs regroupements visibles pour l'aspect en-tête avec un échantillon de 10 000 maliciels. Une couleur est attribuée à chaque famille, c'est-à-dire à chaque étiquette pour faciliter la reconnaissance des groupes lorsque le graphe est visualisé.

Lorsque le nombre de maliciels à regrouper devient grand, le temps et la puissance de calcul requis pour organiser le graphe visuellement augmente. C'est pourquoi nous exportons deux types de graphe pour chaque regroupement. Le premier est le graphe

2. Logiciel utilisé pour visualiser les graphes, https://gephi.org/

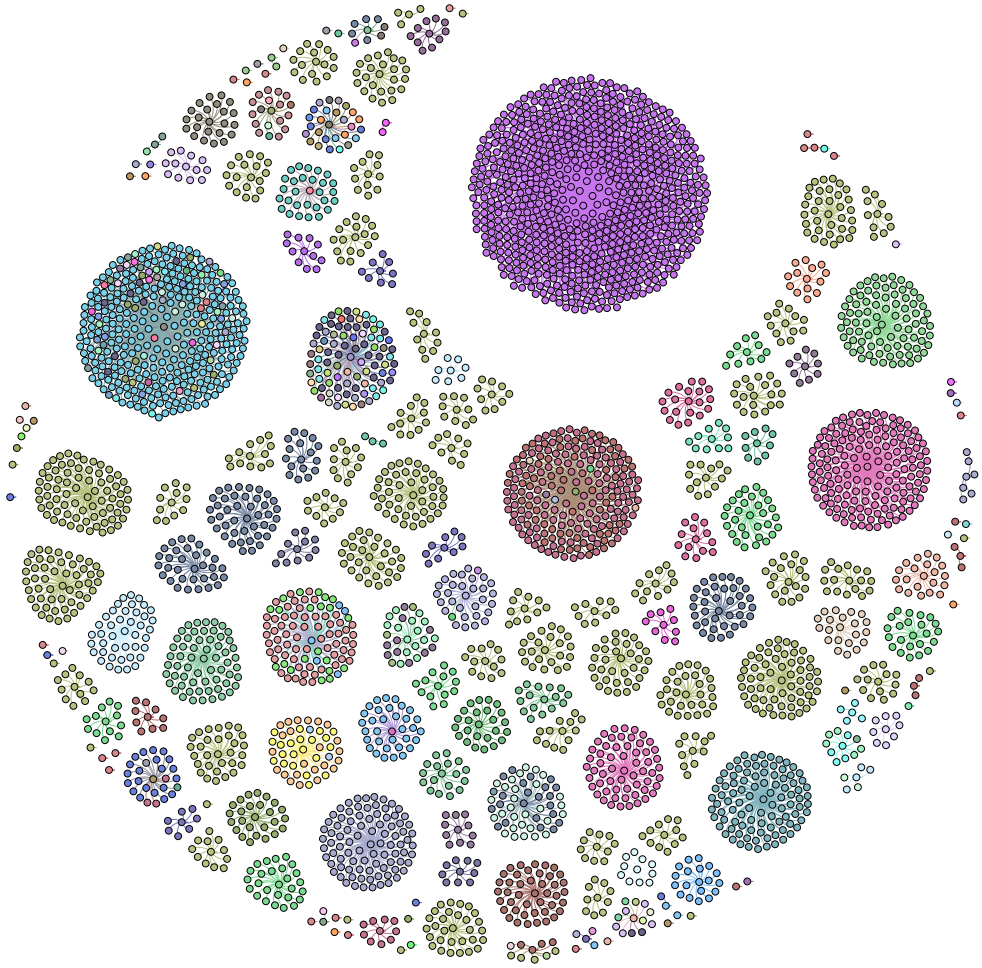


figure 4.2 – Exemple de graphe pour l'aspect en-tête (échantillon de 10 000 maliciels).

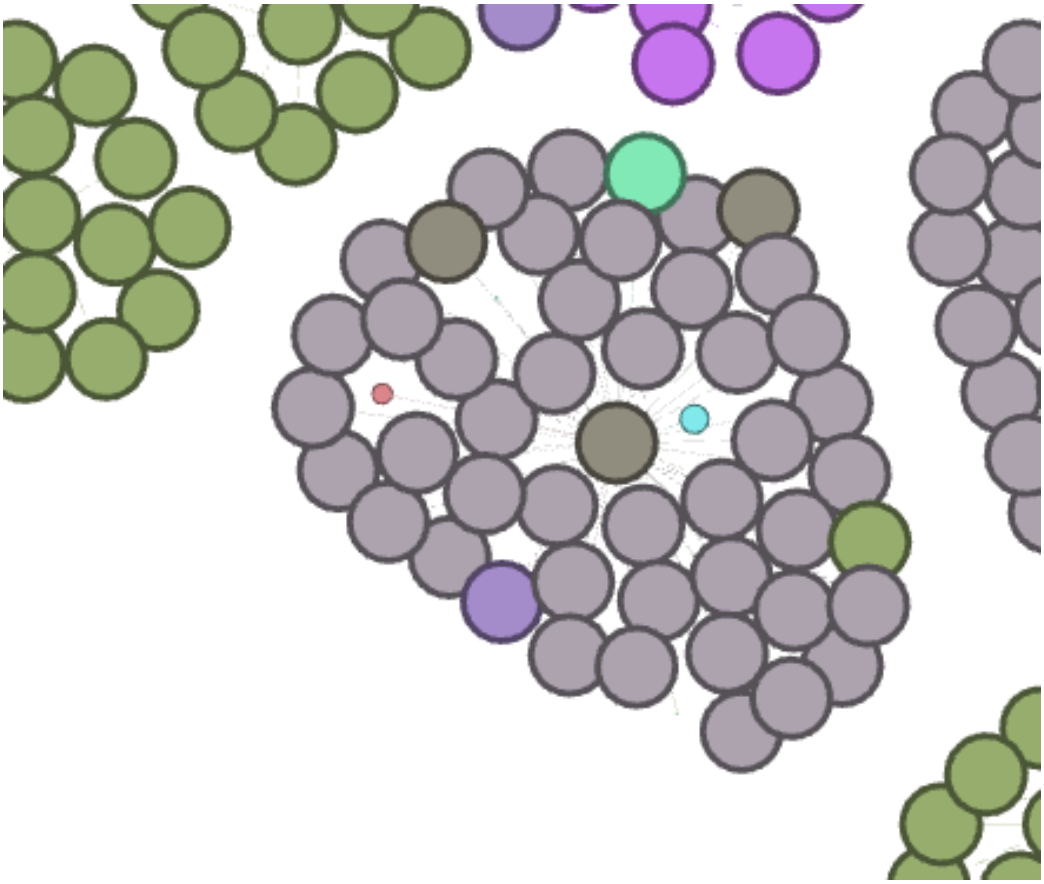
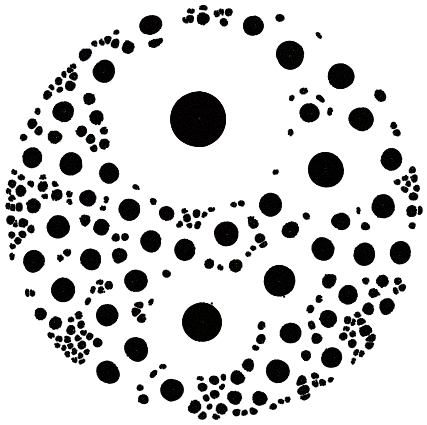


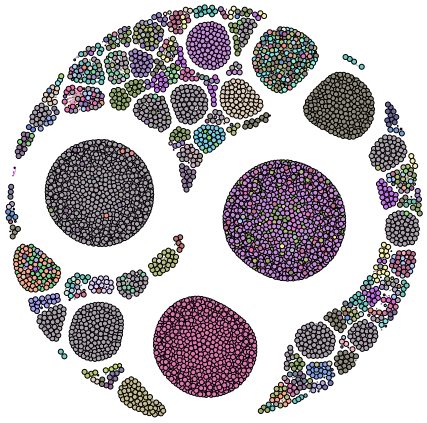
figure 4.3 – Vue proche de différentes tailles de points

standard, où chaque noeud visible correspond à un maliciel regroupé. Le deuxième graphe est une version où jusqu'à 10 maliciels d'une même famille dans un même groupe sont regroupés en un même noeud. De cette façon, plutôt que d'avoir 100 noeuds d'une famille donnée dans un même groupe, seuls 10 noeuds sont affichés, réduisant grandement le nombre d'éléments du graphe. Si moins de 10 maliciels appartiennent à la même famille dans un même groupe, alors un seul noeud est affiché pour cette famille dans ce groupe. La taille du noeud représente alors le nombre de maliciels représentés par le point, comme illustré dans la figure 4.3. La figure 4.4 illustre la comparaison entre les deux types de graphes, avec ou sans rassemblement des points.

Dans cette section, nous avons étudié le programme utilisé pour trouver l'ensemble



(a) Graphe sans rassemblement, comportant tous les points



(b) Graphe avec les points rassemblés 10 par 10

figure 4.4 – Comparaison de graphes avec et sans rassemblement des points pour un échantillon de 50 000

de tests, formater les données pour qu'elles soient utilisables par l'algorithme, exécuter notre algorithme puis générer les mesures, les signatures et le graphe résultant du regroupement. Nous allons à présent décrire les expériences faites pour ajuster les seuils de l'algorithme modifié, ainsi que les résultats de ce dernier.

4.4 Expérimentation et résultats

Les modifications apportées à l'algorithme dans le chapitre 3 ainsi que le format des données utilisées (voir section 4.1) impliquent que des ajustements au niveau des seuils de distances sont nécessaires. Notamment, la distance calculée grâce à l'algorithme modifié n'est pas la même que celle calculée par l'algorithme Malheur original, ce qui veut dire qu'il faut procéder à une recherche de la distance adéquate pour obtenir le meilleur regroupement possible. De plus, les données utilisées sont différentes de celles utilisées originalement et sont sous forme de fichier CSV. Les variables de configuration de l'algorithme comme par exemple la valeur du n pour la formation des n-grammes ou encore les différents seuils doivent être ajustés pour refléter ces différences.

Dans cette partie, nous abordons les expérimentations effectuées avec l'algorithme modifié vu dans le chapitre 3 et ce grâce au programme implémenté et présenté dans la section 4.3. Nous commençons tout d'abord par traiter de la conversion de l'algorithme Malheur en Java dans la sous-section 4.4.1, puis abordons la comparaison des techniques de distance dans la sous-section 4.4.2, suivi des ajustements effectués dans la sous-section 4.4.3. Nous continuons en exposant les différents tests que nous avons effectués pour ajuster les différents seuils dans la sous-section 4.4.4 et abordons l'uniformisation des données dans la sous-section 4.4.5, suivi du processus utilisé pour extraire les signatures dans la sous-section 4.4.6. Enfin, nous finissons par analyser et critiquer les aspects de données choisies dans la sous-section 4.4.7.

4.4.1 Conversion de l'algorithme en Java

CASTOR étant un programme écrit en Java, nous avons réécrit l'algorithme Malheur en Java pour faciliter la modification de ce dernier pour ajouter les variations

à l'algorithme. Lors de cette conversion en Java, nous avons rencontré plusieurs problèmes d'optimisation.

Tout d'abord, il a fallu convertir les vecteurs de caractéristiques en objets Java. Dans l'algorithme original, les vecteurs de caractéristiques sont représentés par deux tableaux à une dimension, un pour la valeur du n-gramme et un pour la fréquence. Ces deux tableaux ont la même taille et la $i^{ème}$ position dans chaque tableau correspond à la valeur ou la fréquence de la $i^{ème}$ caractéristique. L'utilisation d'objets Java demande plus de mémoire, ce qui a causé des problèmes lorsque l'échantillon de maliciels était assez grand, non seulement à cause de l'utilisation de la mémoire mais aussi une trop grande fréquence du ramasse-miettes. Il a alors fallu simplifier les objets représentant les caractéristiques et les vecteurs de caractéristiques en utilisant des objets plus primitifs tels que des `int`, `double` ou encore des `String` plutôt que des objets plus évoluées.

De même, l'objet utilisé pour représenter la matrice de distance lors de la fusion des groupes (voir algorithme 3) a demandé de nombreuses optimisations. En effet, plus le nombre de maliciels à regrouper est grand, plus le nombre de prototypes et le nombre de groupes sont grands, rendant la matrice de distances entre tous les prototypes considérablement plus grande. Nous avons essayé d'utiliser plusieurs bibliothèques disponibles en Java comme par exemple EJML [3], Colt [4], ParallelColt [5] ou encore JBlas [6], cependant, ces bibliothèques ont causé des problèmes de mémoire lorsque l'échantillon devenait grand ou même demandaient un temps de calcul trop long.

Dans un souci de rapidité et d'économie de l'espace mémoire, nous avons essayé de représenter la matrice en un tableau à une dimension. Une matrice de distances est une matrice triangulaire, car $distance_{i,j} = distance_{j,i}$. Par exemple, pour la matrice triangulaire

3. http://ejml.org/wiki/index.php?title=Main_Page
4. https://dst.lbl.gov/ACSSoftware/colt/
5. https://sites.google.com/site/piotrwendykier/software/parallelcolt
6. http://mikiobraun.github.io/jblas/

```
0.0
1.0  5.0
2.0  6.0  9.0
3.0  7.0  10.0  12.0
4.0  8.0  11.0  13.0  14.0
```

le tableau à une dimension équivalent est

```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0]
```

Cette technique de stockage de matrice a grandement amélioré la performance de l'algorithme de regroupement et a aussi permis de regrouper un plus grand nombre d'échantillons. Cependant, il existe une limite de cases contiguës en mémoire qu'il est possible de réserver, posant un problème lorsque le nombre d'échantillons devenait encore plus grand.

Notre troisième essai se rapproche plus d'une matrice triangulaire, tout en utilisant des objets primitifs de Java. En effet, nous avons utilisé un tableau irrégulier (en anglais *jagged array*) à deux dimensions. En Java, il est possible de définir `double[][]` et par la suite de définir la taille de la deuxième dimension, qui peut être différente pour chaque ligne. La matrice triangulaire

```
0.0
1.0  5.0
2.0  6.0  9.0
3.0  7.0  10.0  12.0
4.0  8.0  11.0  13.0  14.0
```

est donc représentée en Java par le tableau à deux dimensions

```
[[0.0], [1.0, 5.0], [2.0, 6.0, 9.0], [3.0, 7.0, 10.0, 12.0], [4.0, 8.0, 11.0, 13.0, 14.0]]
```

Cette représentation est assez rapide et assez peu gourmande en mémoire pour permettre des regroupements avec de nombreux échantillons.

4.4.2 Différents calculs de distance

L'algorithme original de Malheur calcule les caractéristiques en appliquant la fonction de hachage MD5 pour chaque n-gramme. Cette fonction de hachage a le désavantage que le moindre changement au texte en entrée produit un MD5 complètement

tableau 4.10 – Durée moyenne du calcul de distance entre deux vecteurs de caractéristiques

Variation	Durée moyenne (en microsecondes)
Original	2,5
Dissimilitude cosinus	46,5
SSDeep	552,3

différent. Cependant, utiliser cette technique apporte un gain non seulement au niveau de l'espace mémoire utilisé, mais également un gain de temps lors du calcul des distances. En effet, le tableau 4.10 montre que la durée moyenne nécessaire pour calculer la distance entre deux vecteurs de caractéristiques pour cette technique est de 2,5 microsecondes seulement. Ce faible temps de calcul permet ainsi au regroupement de s'effectuer en un temps raisonnable.

Le tableau 4.10 montre aussi que le temps de calcul des variations de distances implémentées est 18 à plus de 200 fois plus long. Nous avons remarqué lors de nos expérimentations que pour 50 000 maliciels à regrouper, la variation avec SSDeep peut prendre plus de 15 heures, tandis que la version originale ou même celle utilisant la dissimilitude cosinus prennent moins de 3 heures. Une telle durée de regroupement pose problème lorsque le nombre d'échantillons continue à grandir.

Pour minimiser le temps de calcul de distance avec la version utilisant la recherche approximative, nous avons essayé plusieurs algorithmes, notamment MinHash, FuzzyWuzzy [7], LSH et SSDeep. Nos tests ont montré que SSDeep est le plus rapide pour calculer la distance entre deux vecteurs de caractéristiques. Cependant, cette durée de calcul reste importante et peut devenir un problème majeur lorsque l'échantillon utilisé grandit.

4.4.3 Ajustement de la taille des *n*-grammes

Plusieurs variables affectent l'algorithme de regroupement présenté dans la section 3.2. Tout d'abord, la taille du n-gramme définit la taille de la fenêtre coulissante, ce qui veut dire que plus le n est grand, plus la séquence d'instructions formant le n-gramme est grande. La probabilité qu'une séquence de deux instructions se retrouve

7. http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

dans deux maliciels de familles différentes est plus grande que la probabilité qu'une séquence de quatre instructions se retrouve dans ces mêmes maliciels. En somme, plus la taille de la séquence est importante, plus la séquence risque d'être unique et plus le regroupement devient précis. Il faut cependant trouver la bonne taille de séquence pour que le regroupement puisse former des groupes qui englobent toute une famille sans être trop permissif.

Le choix de la taille des n-grammes est souvent une question importante lors de regroupements. En effet, la taille a un impact non seulement sur le regroupement, mais aussi sur les seuils de distances entre les prototypes, entre un élément et son prototype ou entre les groupes. Plus la taille des n-grammes est importante, plus les distances entre les éléments sont importantes, car il est moins probable que les séquences de n lignes se retrouvent dans plusieurs maliciels. Cela implique donc que plus la taille des n-grammes est importante, plus les seuils de distances doivent être importants.

Le n est fourni par un fichier de configuration et par défaut est égal à 2. Pour Hu et Tan [8], une valeur de $n = 4$ pour des n-grammes est habituellement une valeur qui permet de regrouper de façon robuste sans perdre en performance. Dans nos expériences, nous avons tenté de déterminer la valeur permettant le meilleur regroupement. Pour ce faire, nous avons effectué des regroupements avec des n-grammes de tailles 2, 3 ou 4 et nous avons comparé la qualité des regroupements grâce aux mesures internes et externes calculées par notre programme (voir section 4.3). Lors de nos expériences, nous avons rencontré des problèmes de performance plus le n devenait grand. En effet, une valeur de 4 pour le n pour l'aspect hôte avec l'algorithme utilisant la dissimilitude cosinus a demandé tant de mémoire que nous n'avons pas pu compléter le regroupement. De même, un n de 5 ou plus allonge le temps d'exécution de façon importante et demande beaucoup de mémoire, ce qui pose problème lorsque l'ensemble de données est grand.

Nous pouvons remarquer grâce à la figure 4.5 qu'avec un n de taille 3 ou 4, les groupes sont plus précis. En effet, tel que mentionné précédemment, la taille de la séquence d'instructions a une influence sur la probabilité qu'un autre maliciel ait la même séquence d'instructions même s'il n'est pas de la même famille. Une valeur de 3 ou 4 pour le n selon la mesure de distance semble être un bon compromis entre une

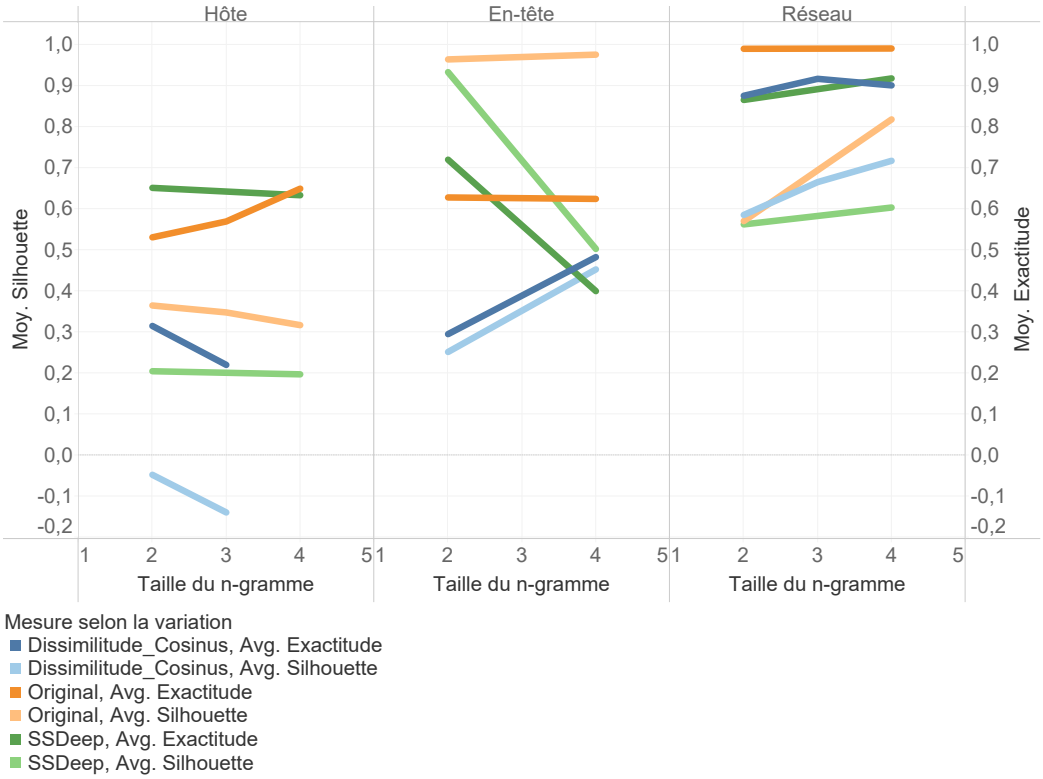


figure 4.5 – Variation de l'indice Silhouette et de l'exactitude par rapport à la taille du n-gramme pour chaque aspect et chaque variation

taille de séquence d'instructions trop courte et trop longue.

4.4.4 Ajustement des seuils

Il existe différentes valeurs utilisées dans l'algorithme qui peuvent être configurés pour obtenir un meilleur regroupement. Ces valeurs sont des seuils de distance au dessus ou en dessous desquels la décision à prendre change. Par exemple, si la distance entre un élément et son prototype est en dessous d'un certain seuil, alors cet élément est assigné au groupe représenté par le prototype. Les trois seuils de distances sont les suivants :

— S_1, le seuil de distance maximale entre deux prototypes (voir algorithme 2) ;
— S_2, le seuil de distance minimale entre deux groupes (voir algorithme 3) ;
— S_3, le seuil de distance maximale entre un vecteur et son prototype (voir algo-

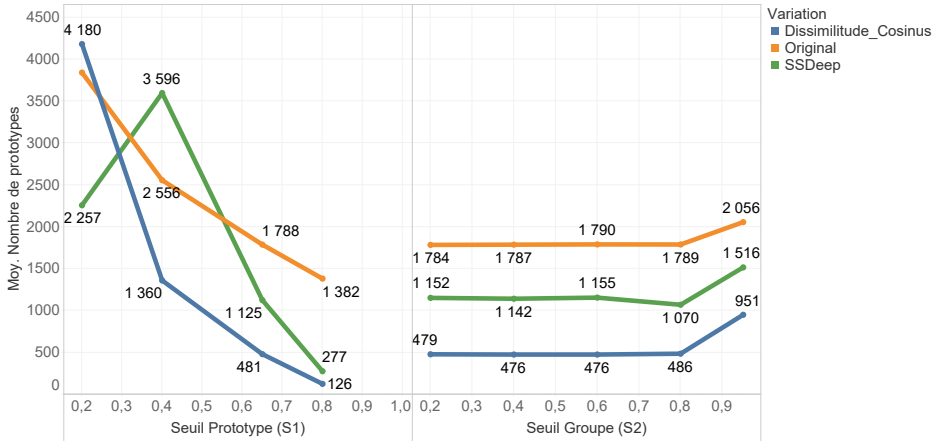
rithme 4).

Ces valeurs ont une influence sur le nombre de prototypes choisis, le nombre de groupes résultants et sur les éléments qui forment chaque groupe. La variation apportée à l'algorithme pour utiliser la recherche approximative a une influence sur la valeur des seuils qui produit le meilleur regroupement, qui doivent aussi être ajustés pour produire le meilleur regroupement possible.

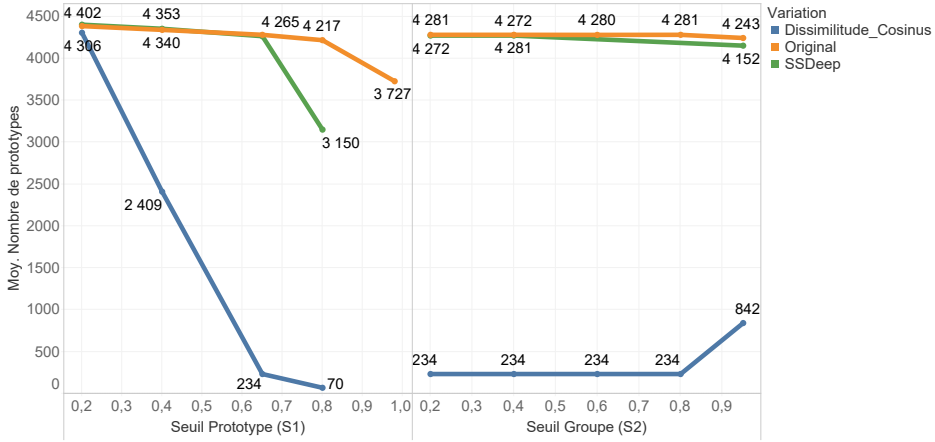
Les figures 4.6c, 4.6a et 4.6b montrent la variation du nombre de prototypes en fonction de l'ajustement du seuil S_1. Il est alors possible de remarquer que si S_1 est assez grand, alors le nombre des prototypes extraits est petit, ce qui résulte en moins de groupes, ces derniers étant plus gros. De même, si S_2 est assez grand, plus de groupes sont fusionnés ensemble, résultant en moins de groupes car ceux-ci ont été davantage fusionnés ensemble. Ceci permet par exemple de fusionner les groupes de maliciels de même famille mais de campagnes différentes (voir sous-section 1.1.2.2), délivrant ainsi un meilleur regroupement. Enfin, diminuer la valeur de S_3 revient à exiger des groupes pour lesquels les éléments sont plus proches les uns des autres, qualité importante d'un bon regroupement.

Les figures 4.7a, 4.7b et 4.7c illustrent la variation de l'indice Silhouette et de l'exactitude en fonction de l'ajustement des trois différents seuils. Ces figures montrent que l'ajustement du seuil dépend de l'aspect mais aussi de la mesure. En effet, pour la variation utilisant la dissimilitude cosinus, un seuil S_1 autour de 0,2 donne de meilleurs résultats, tandis qu'un seuil d'environ 0,4 génère de meilleurs résultats pour la variation Malheur SSDeep et Malheur original. De même, nous remarquons que le seuil S_2 donne de meilleurs résultats s'il est autour de 0,9, sauf pour l'aspect réseau, où une valeur un peu plus faible (0,8) semble mieux fonctionner. Enfin, S_3 semble atteindre sa valeur idéale autour de 0,7 et 0,9. Les tableaux A.1 et A.2 contiennent les résultats complets des expériences réalisées pour déterminer les seuils les plus appropriés et réaliser ces figures.

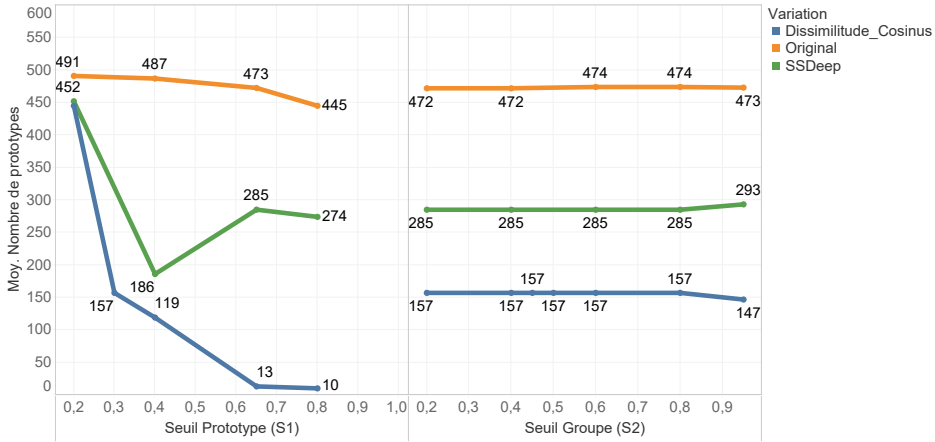
Les figures 4.7a, 4.7b et 4.7c montrent aussi que la variation utilisant l'algorithme original semble obtenir de meilleurs résultats quel que soit l'aspect. Nous pouvons cependant remarquer grâce aux figues 4.8a, 4.8b et 4.8c que même si l'exactitude et l'indice Silhouette sont élevés, cela n'est pas nécessairement équivalent à regroupement idéal. En effet, la ligne de référence dans ces trois figures indique le nombre de



(a) Variation du nombre de prototypes pour l'aspect hôte (échantillon = 50 000)

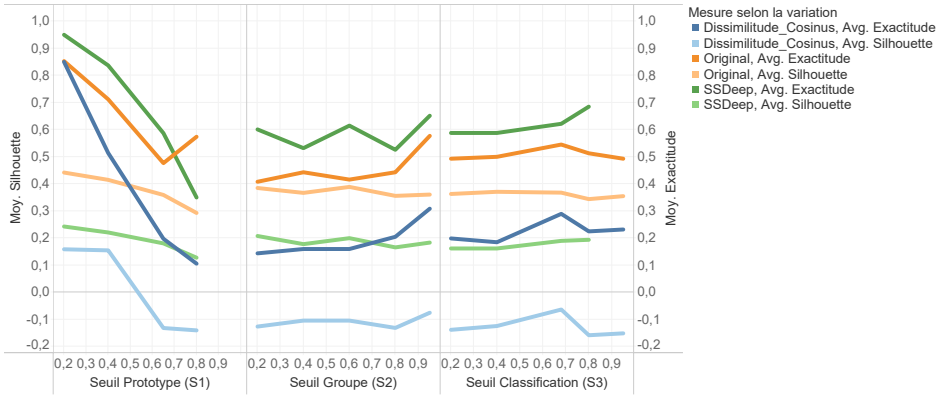


(b) Variation du nombre de prototypes pour l'aspect en-tête (échantillon = 50 000)

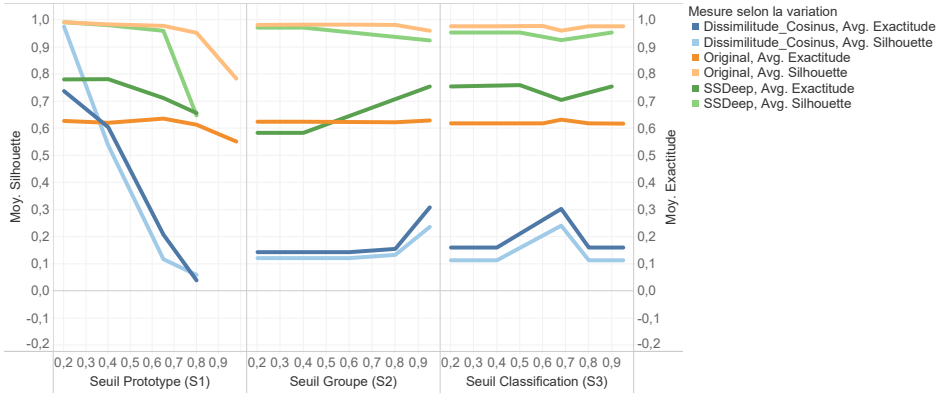


(c) Variation du nombre de prototypes pour l'aspect réseau (échantillon = 500)

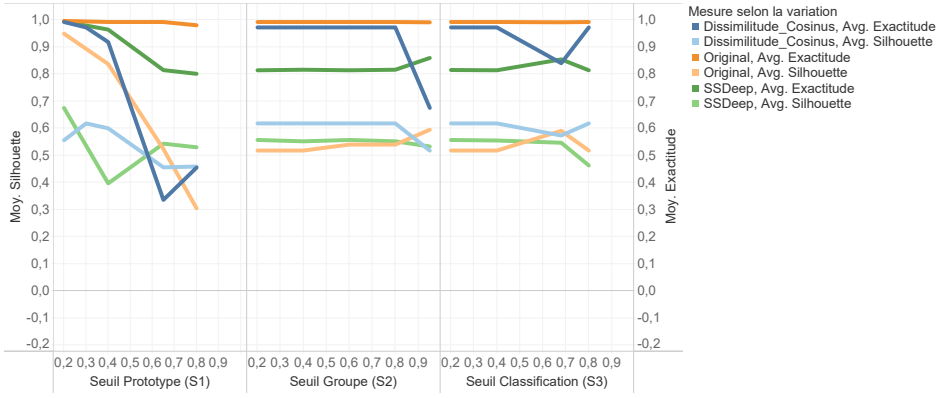
figure 4.6 – Variation du nombre de prototypes pour les trois aspects (n=2)



(a) Variation de l'indice Silhouette et de l'exactitude en fonction des seuils pour l'aspect hôte (échantillon = 50 000)



(b) Variation de l'indice Silhouette et de l'exactitude en fonction des seuils pour l'aspect en-tête (échantillon = 50 000)



(c) Variation de l'indice Silhouette et de l'exactitude en fonction des seuils pour l'aspect réseau (échantillon = 500)

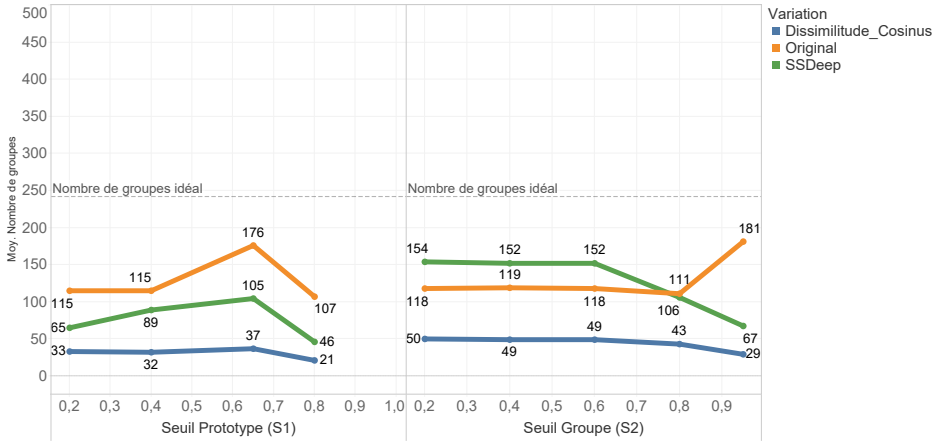
figure 4.7 – Variation de l'indice Silhouette et de l'exactitude en fonction des seuils pour les trois aspects (n=2)

tableau 4.11 – Valeurs utilisées qui ont donné le meilleur regroupement pour chaque aspect et variation

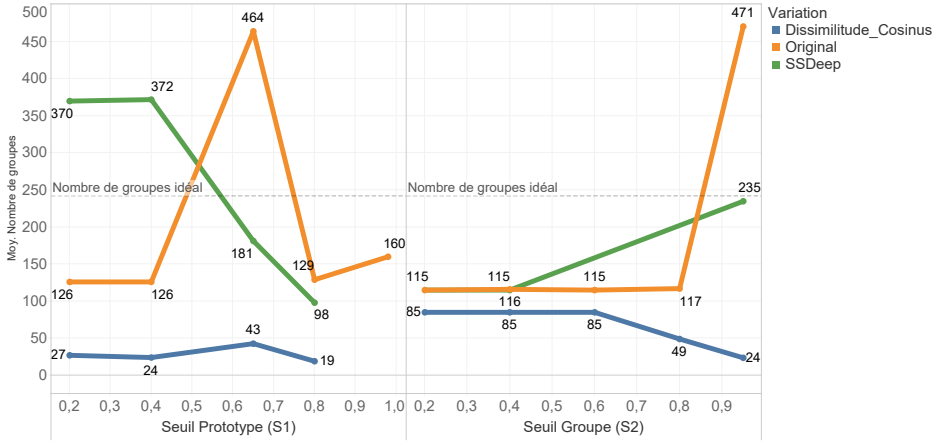
Aspect	Variation	Nombre de groupes	Seuil Prototype (S1)	Seuil Groupe (S2)	Seuil Classification (S3)
Hôte	Dissimilitude Cosinus	109	0.20	0.60	0.90
	Original	113	0.65	0.95	0.68
	SSDeep	471	0.10	0.80	0.80
En-tête	Dissimilitude Cosinus	178	0.30	0.50	0.80
	Original	127	0.65	0.95	0.68
	SSDeep	115	0.65	0.40	0.68
Réseau	Dissimilitude Cosinus	110	0.30	0.50	0.80
	Original	423	0.65	0.95	0.68
	SSDeep	278	0.10	0.80	0.80

groupes attendus selon les étiquettes présentes dans l'échantillon de test (250 familles pour l'aspect hôte et en-tête, 200 pour l'aspect réseau). Le nombre de familles attendu est déterminé grâce au nombre d'étiquettes différentes disponibles pour chaque ensemble de test. Ces figures montrent que la technique utilisée par l'algorithme Malheur original permet des groupes plus nombreux mais plus homogènes, tandis que celles utilisant la recherche approximative (SSDeep ou dissimilitude cosinus) produisent moins de groupes, mais ces derniers sont moins précis. En effet, l'utilisation de la fonction de hachage MD5 pour calculer les caractéristiques permet de rapprocher des caractéristiques identiques et mais ne permet pas d'évaluer la distance entre des caractéristiques non identiques. Au contraire, l'utilisation de la recherche approximative lors du regroupement permet d'estimer le degré de similitude entre deux caractéristiques, qu'elles soient identiques ou non. Cela résulte en des distances moins élevées entre les différentes caractéristiques, menant à un regroupement moins homogène, présentant ainsi des mesures moins élevées. Visualiser les graphes correspondant permet cependant de remarquer cette particularité, comme illustré par la figure 4.9, où il est clairement visible que l'algorithme original produit des groupes plus homogènes mais plus petits, tandis que les variations utilisant SSDeep ou la dissimilitude cosinus produisent moins de groupes et ceux-ci sont moins homogènes.

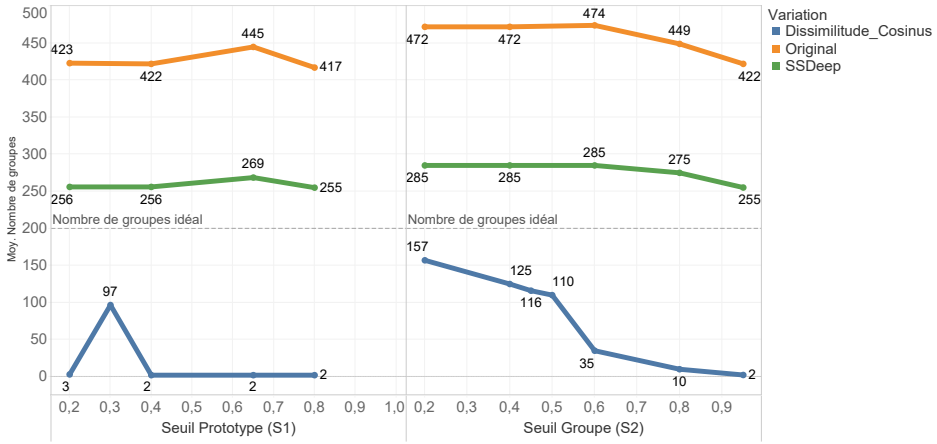
Grâce aux informations récoltées dans les expériences et tableaux précédents, nous avons déterminé les valeurs des seuils selon la variation de l'algorithme pour chaque aspect étudié. Le tableau 4.11 présente ces seuils ainsi que le nombre de groupes résultant d'un regroupement effectué avec ces seuils. Les graphes générés lors de ces regroupements sont inclus dans la section A.2.



(a) Variation du nombre de groupes pour l'aspect hôte (échantillon = 50 000, nombre de familles attendues = 250)

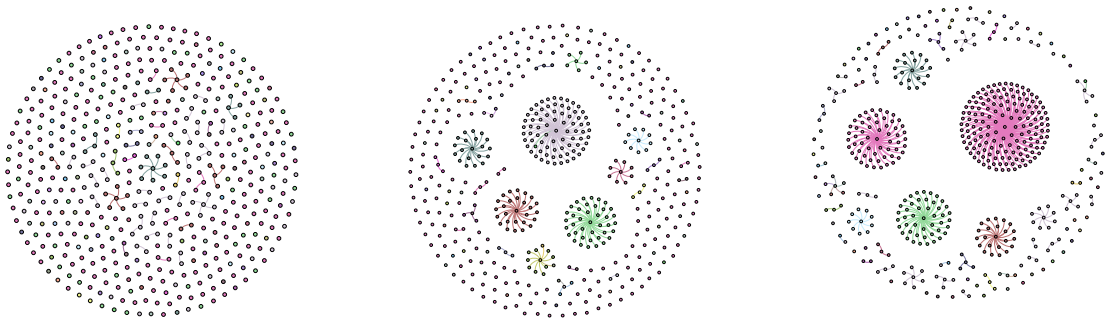


(b) Variation du nombre de groupes pour l'aspect en-tête (échantillon = 50 000, nombre de familles attendues = 250)



(c) Variation du nombre de groupes pour l'aspect réseau (échantillon = 500, nombre de familles attendues = 200)

figure 4.8 – Variation du nombre de groupes pour les trois aspects (n=2)



(a) Graphe généré avec l'algorithme original pour l'aspect réseau

(b) Graphe généré avec l'algorithme utilisant SSDeep pour l'aspect réseau

(c) Graphe généré avec l'algorithme utilisant la dissimilitude cosinus pour l'aspect réseau

figure 4.9 – Comparaison des groupes générés par les différentes variations de l'algorithme pour l'aspect réseau (échantillon = 500)

4.4.5 Uniformisation des données

Les données utilisées comme entrée du regroupement sont des données textes, provenant de traces d'exécution de maliciels. Cependant, ces derniers utilisent différentes techniques comme par exemple l'obscurcissement (voir sous-section 1.1.2.2) pour éviter d'être reconnus par des logiciels d'antivirus. L'utilisation d'un nom de fichier contenant un nombre aléatoire ou la date et l'heure au moment de la création du fichier sont notamment des techniques utilisées par les maliciels pour empêcher la reconnaissance facile de leur exécution. Ces techniques permettent donc à deux exécutions d'un même maliciel de ne pas avoir la même trace d'exécution. Ces différences entre maliciels identiques peuvent faire en sorte que le regroupement ne puisse pas regrouper ensemble de tels maliciels.

Pour remédier à ce problème, nous avons procédé à une uniformisation des données en enlevant les parties des données qui seraient différentes pour chaque exécution. Par exemple, nous avons remplacé le nom de l'exécutable par *SUSPICIOUS.EXE* au lieu de son MD5, pour que le MD5 du maliciel n'ait pas d'influence sur le regroupement. En effet, les auteurs de maliciels peuvent utiliser des obscurcisseurs pour faire en

sorte que le même maliciel ait un MD5 différent à chaque fois qu'il est généré (voir section 1.1.2.2). De même, les dates et valeurs aléatoires, c'est-à-dire les chaînes de caractères respectant le format `yyyy-MM-dd` et les valeurs numériques de plus de 6 chiffres ont été remplacées par une valeur fixe pour que les valeurs dépendant du moment d'exécution n'aient pas d'influence sur le regroupement.

Lors de nos expériences, nous avons pu remarquer que l'uniformisation des données a un impact négligeable sur le regroupement. Cela peut s'expliquer par le fait qu'il est difficile d'uniformiser toutes les valeurs changeantes. En effet, il existe une liste infinie de valeurs à uniformiser, comme par exemple :

— les nombres aléatoires ;
— les dates aléatoires ;
— les noms de fichiers aléatoires ;
— les séquences d'octets ;
— les chaînes de caractères aléatoires.

De plus, même en établissant une liste exhaustive de toutes les valeurs à uniformiser, celle-ci ne serait valable que pour les données de l'échantillon, puisqu'un autre échantillon posséderait d'autres valeurs à uniformiser. De même, l'écosystème des maliciels étant en constante évolution, avec de nouveaux maliciels apparaissant chaque jour. Former une liste exhaustive des valeurs à uniformiser demande un travail constant et n'est pas assez dynamique. Une façon de détecter automatiquement les valeurs à uniformiser ou d'utiliser une méthode tolérante aux valeurs changeantes est nécessaire.

Il n'est pas non plus possible de savoir quelles valeurs devraient être uniformisées ou non. Par exemple, une chaîne de caractères telle que

« `user_id=D3A67B90-4D23-415E-94BC-C71E930A2606` »

peut être considérée comme aléatoire et uniformisée, même s'il s'avère qu'elle ne change pas à travers les résultats d'analyses de maliciels et serait ainsi une très bonne caractéristique pour regrouper des maliciels. De même, la chaîne de caractères présente dans un appel réseau décrivant l'agent utilisateur (en anglais *user agent*) peut être considérée comme un détail anodin et donc uniformisé pour que le navigateur utilisé lors de l'exécution du maliciel n'influence pas le regroupement. Cependant, certains maliciels utilisent l'agent utilisateur « `Mozilla/4.08 (Charon; Inferno)` », caractéristique qui pourrait permettre de mieux regrouper certains maliciels.

En somme, l'uniformisation des données doit se cantonner aux valeurs qui sont différentes pour chaque maliciel ou exécution, sans possibilité d'être importante ou assez spécifique pour être utilisée comme caractéristique discriminante lors du regroupement. Dans le jeu de données du CCRIC, le maliciel analysé est toujours nommé d'après son MD5, rendant l'uniformisation pour cette valeur utile. En effet, le nom original du fichier n'est pas présent et tous les maliciels sont nommés de la même façon d'après leur MD5 par le CCRIC. Uniformiser cette valeur est alors sans risque et ne peut avoir qu'un impact positif sur le regroupement.

4.4.6 Étude des signatures extraites

Les signatures extraites sont choisies en fonction de la fréquence des instructions pour tous les éléments d'un même groupe. Le résultat de l'extraction est présenté sous la forme d'un fichier CSV d'un format similaire à celui des fichiers CSV utilisés en entrée (voir le tableau 4.4 pour l'aspect hôte, le tableau 4.6 pour l'aspect en-tête et le tableau 4.8 pour l'aspect réseau).

Cette façon d'extraire les signatures implique donc que l'ordre dans lequel les instructions ont été effectuées est perdu. Garder la relation entre les instructions serait intéressant, cependant il est difficile de garder cette relation pour tous les éléments du groupe. En effet, pour certains éléments du groupe, une instruction A et B peuvent se suivre tandis que pour d'autres, les instructions A et C se suivent. Il est donc difficile de conserver l'ordre des instructions lorsque les instructions A, B et C sont sélectionnées pour faire partie de la signature.

En inspectant les signatures générées, nous avons pu remarquer plusieurs problèmes :

— certaines signatures ne possèdent pas d'instructions assez spécifiques ;

— certains groupes avec beaucoup d'éléments n'ont pas de signatures.

Tout d'abord, certaines instructions telles que `GetSystemDirectoryW` ou encore les requêtes pour obtenir le nom de l'ordinateur pour l'aspect hôte sont souvent incluses dans la signature pour de nombreux groupes. Ces deux instructions sont des commandes communes, que ce soit pour un maliciel ou non. Pour empêcher les instructions trop générales d'être sélectionnées comme signatures, il faudrait mettre en

place un système de filtre, gardant ainsi uniquement les instructions assez spécifiques pour correspondre aux éléments du groupe et à un minimum des éléments des autres groupes.

De plus, plusieurs signatures extraites sont vides et ce, malgré le fait que le groupe possède de nombreux éléments. En effet, en introduisant l'utilisation de recherche approximative pour former les groupes dans l'algorithme de regroupement, les maliciels aux instructions à peu près égales peuvent être mis dans un même groupe. Cependant, lors de l'extraction de la signature du groupe, aucune recherche approximative n'est utilisée pour trouver la fréquence des instructions. Mieux encore, modifier l'instruction en ajoutant une expression régulière pour qu'elle corresponde à un plus grand nombre d'éléments pourrait être bénéfique. Par exemple, une signature extraite telle que

```
POST /vtapi/v2/file/scan HTTP/1.1 Content-Type: multipart/form-data; boundary=------000000000205515 User-Agent: [...]
POST /vtapi/v2/file/scan HTTP/1.1 Content-Type: multipart/form-data; boundary=------000000000236406 User-Agent: [...]
POST /vtapi/v2/file/scan HTTP/1.1 Content-Type: multipart/form-data; boundary=------000000000250484 User-Agent: [...]
POST /vtapi/v2/file/scan HTTP/1.1 Content-Type: multipart/form-data; boundary=------000000000210781 User-Agent: [...]
POST /vtapi/v2/file/scan HTTP/1.1 Content-Type: multipart/form-data; boundary=------000000000241390 User-Agent: [...]
POST /vtapi/v2/file/scan HTTP/1.1 Content-Type: multipart/form-data; boundary=------000000000219781 User-Agent: [...]
```

pourrait être simplifiée et généralisée en

```
POST /vtapi/v2/file/scan HTTP/1.1 Content-Type: multipart/form-data; boundary=-----(\d+) User-Agent: [...].
```

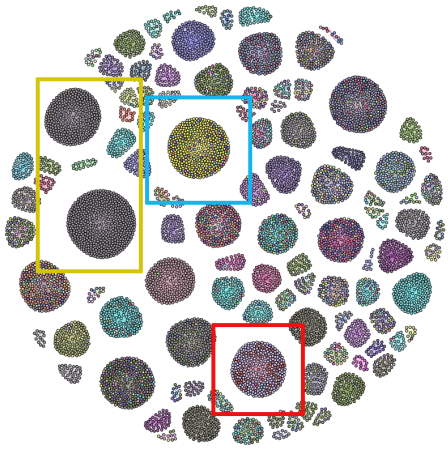
Ainsi, la signature est simplifiée et peut aussi correspondre à un plus grand nombre d'éléments.

4.4.7 Étude des différents aspects

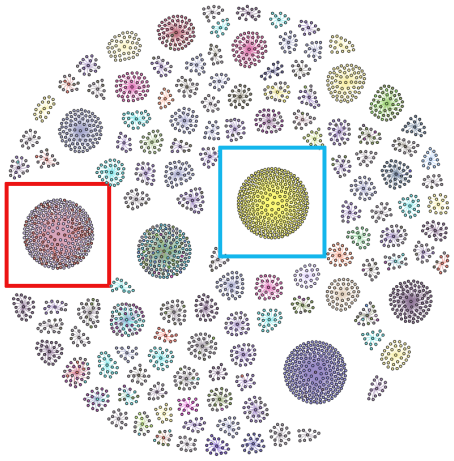
Nous avons choisi plusieurs aspects de données pour effectuer le regroupement. Tout d'abord, nous avons étudié les actions du maliciel au niveau de l'hôte grâce à des fichiers contenant des résultats d'analyses dynamiques enregistrées au niveau de l'hôte, ceux-ci étant organisés de la façon suivante : nom du processus source, action (e.g., sur un dossier, sur une clé de registre), valeur associée à l'action (e.g., chemin jusqu'au dossier ou la clé de registre) (voir sous-section 4.1.1). Le deuxième aspect est celui qui utilise des fichiers contenant des résultats d'analyses d'en-têtes de fichiers, ceux-ci étant organisés en trois colonnes : le nom de la bibliothèque importée, le nom de la méthode utilisée et l'adresse du point d'entrée (voir sous-section 4.1.2). Le

troisième et dernier aspect est celui relié aux actions du maliciel au niveau des appels réseaux grâce à des fichiers contenant des résultats d'analyses de données réseau. Ces fichiers sont organisés en trois colonnes, avec en premier le domaine ou l'adresse IP que le maliciel a essayé de rejoindre, puis le protocole utilisé suivi du port et enfin les données envoyées lors de l'appel réseau (voir sous-section 4.1.3)

Nous avons effectué des expérimentations avec ces différents aspects pour essayer de déterminer si un aspect est meilleur lorsqu'il s'agit de regrouper un maliciel selon son comportement. Les graphes présents dans la section A.2 permettent de remarquer que les trois aspects étudiés produisent des groupes à peu près homogènes et de taille raisonnable. Ces aspects semblent donc être assez caractéristiques pour regrouper les maliciels en familles de façon correcte.



(a) Comparaison des groupes générés lors de regroupement sur l'aspect hôte



(b) Comparaison des groupes générés lors de regroupement sur l'aspect en-tête

figure 4.10 – Comparaison des groupes générés par l'aspect hôte et en-tête

Nous remarquons grâce à la figure A.9 que l'aspect en-tête produit des groupes plus homogènes que l'aspect hôte. Cependant, il est aussi possible de remarquer que la même famille de maliciels est représentée dans plusieurs groupes, indiquant que l'aspect en-tête ne permet pas à lui seul d'effectuer un regroupement parfait des maliciels. Les aspects hôte et réseau semblent aussi souffrir de ce problème comme illustré par figure A.6 et figure A.3, même si l'aspect hôte semble un peu moins affecté et possède des groupes plus gros. En effet, le comportement au niveau de

l'hôte change d'un maliciel à un autre et ce, même s'ils appartiennent au même type de maliciels. Par exemple, un rançongiciel va tout d'abord demander à son C2 la clé qu'il doit utiliser pour effectuer l'encryption, puis lister les fichiers et commencer à encrypter, tandis qu'un autre génère une clé, encrypte et envoie ensuite la clé générée à son C2. Ces deux maliciels sont alors différenciables par leurs séquences d'actions. De même, un maliciel non reconnu par les antivirus et non étiquetable qui possède ce comportement peut facilement être regroupé avec précision grâce aux séquences d'actions qu'il effectue.

La figure figure 4.10 permet de remarquer que malgré les différences entre les aspects en-tête et hôte, ces derniers possèdent certaines ressemblances. En effet, les groupes encadrés en rouge semblent contenir les mêmes éléments d'un aspect à l'autre. Les groupes encadrés en bleu semblent au contraire être plus homogènes dans le graphe représentant le regroupement sur l'aspect en-tête. De même, les groupes encadrés en jaune n'apparaissent que dans le graphe représentant le regroupement sur l'aspect hôte et sont dispersés en une multitude de petits groupes dans le graphe représentant le regroupement effectué sur l'aspect en-tête.

Les groupes encadrés en rouge dans la figure 4.10 sont également remarquables car dans deux des aspects étudiés (en-tête et hôte), ces maliciels forment un groupe. Il est possible de voir dans la figure 4.11 que ces maliciels appartiennent aux familles «*klez*» et «*sytro*». La signature extraite de ce groupe possède 206 instructions pour un groupe de plus de 300 éléments, indiquant un bon regroupement. En étudiant les instructions présentes dans la signature, nous pouvons remarquer que beaucoup d'entre elles sont en réalité des créations de fichiers avec des noms spécifiques pour encourager l'utilisateur à exécuter le maliciel. Un extrait du fichier de la signature pour ce groupe est visible ci-dessous :

```
SUSPICIOUS.exe,file_created,C:\WINDOWS\Temp\Key generator for all windows XP versions.exe
SUSPICIOUS.exe,file_date_change,C:\WINDOWS\Temp\AikaQuest3Hentai FullDownloader.exe
SUSPICIOUS.exe,file_created,C:\WINDOWS\Temp\Hack into any computer!!.exe
```

Ce groupe à priori hétérogène semble en fait regrouper des maliciels d'une même famille qui possède deux noms différents. Cela veut donc dire que «*klez*» et «*sytro*» sont des variantes et que le groupe a été correctement regroupé.

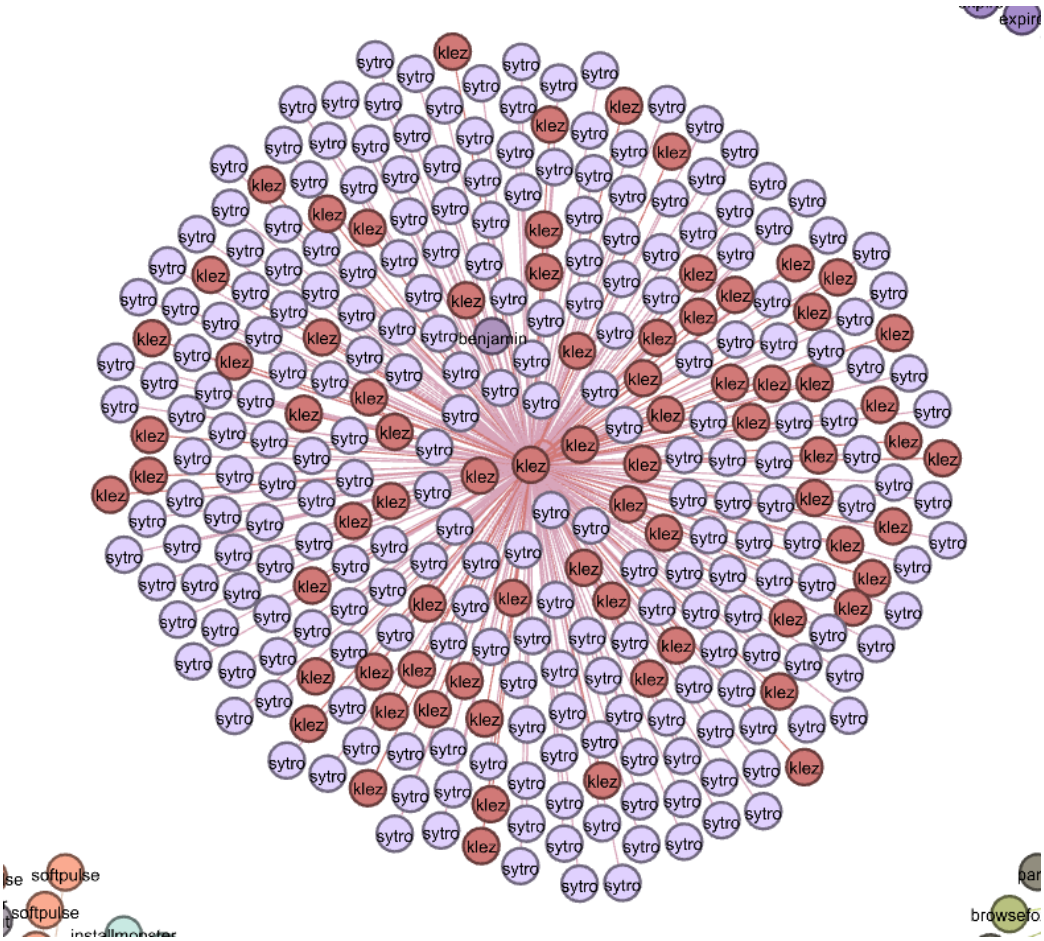


figure 4.11 – Maliciels de deux familles différentes regroupés dans un même groupe de façon consistante

Conclusion

Dans cette section, nous avons pu étudier en détail l'impact des différents seuils qui entrent en jeu dans l'algorithme de regroupement. Ces seuils ont une influence sur la qualité du regroupement final et le bon ajustement de ces derniers permet d'avoir un regroupement ayant des groupes plus homogènes. De même, nous avons comparé l'algorithme Malheur original avec nos deux variations de l'algorithme, soit Malheur SSDeep et Malheur dissimilitude cosinus. L'utilisation de la recherche approximative apporte une amélioration au regroupement en générant des groupes plus gros, quoiqu'un peu moins homogènes. Cependant, le calcul de distance avec cette technique demande plus de temps, ce qui pose problème lorsque l'échantillon de maliciels à regrouper devient grand.

Nous avons de plus analysé les différents aspects de données utilisés et il nous a été possible d'estimer que malgré le fait que les aspects choisis génèrent de bon regroupements, un aspect combiné résulterait sûrement en un meilleur regroupement. En effet, plus les séquences d'instructions présentes dans les fichiers CSV en entrée sont précises et différentes, meilleur est le regroupement. Cependant, le regroupement actuel permet déjà à un analyste de découvrir des faits intéressants tel que la détection de variantes ou les instructions les plus fréquentes dans un groupe et ce de façon semi-automatique.

Conclusion

Ce mémoire s'intéresse au regroupement de maliciels en utilisant une version modifiée de l'algorithme Malheur [17]. Cet algorithme permet de regrouper un grand nombre de données en un temps d'exécution de $O(k^2 \log k + n)$, où k et n sont respectivement le nombre de prototypes et le nombre d'éléments et ce grâce à la façon dont les vecteurs de caractéristiques sont formés.

L'algorithme Malheur utilise des données traitées et généralisées pour permettre l'utilisation de caractéristiques représentées par une empreinte. L'utilisation d'une telle technique sous-entend que le calcul de la distance entre les caractéristiques ne permet pas de variation mineure de cette dernière. Cependant, cette approche ne fonctionne que lorsque les données sont traitées à l'avance ou qu'il est nécessaire de regrouper des valeurs qui ne possèdent pas de variations mineures ou de valeurs aléatoires. En effet, le traitement de données peut éliminer les valeurs comme par exemple des dates ou des éléments spécifiques qui ne devraient pas avoir un impact sur le regroupement. Ces éléments peuvent donc être normalisés pour ne pas biaiser le regroupement avec des valeurs qui seraient toujours différentes.

L'uniformisation des données est cependant difficile et délicate puisqu'une date ou une valeur aléatoire dans un cas peut être spécifique et fixe dans un autre, rendant impossible l'uniformisation parfaite des données. L'utilisation de la recherche approximative pour calculer la distance entre les caractéristiques permet de regrouper ensemble des données de type texte pas nécessairement traitées et ce, même si elles ne sont pas complètement identiques. La recherche approximative est une technique qui compare les n-grammes partie par partie sur un texte donné, ce qui rend possible la comparaison approximative. Le regroupement de valeur similaires est donc atteignable grâce à cette technique et peut être applicable à de nombreuses données

textuelles. Nous avons cependant pu remarquer grâce aux expériences effectuées, que l'utilisation de la recherche approximative permet un bon regroupement, avec des groupes relativement homogènes pourvu qu'on utilise les bons ajustements.

Les expériences réalisées nous ont permis de trouver les valeurs à fournir à l'algorithme de regroupement selon la variation de l'algorithme utilisée pour optimiser ce dernier. En effet, nous avons effectué une série de tests et de réglages pour accommoder les changements apportés au calcul de la distance selon la variation de l'algorithme utilisé. Ces tests nous permettent de déterminer les meilleurs seuils ainsi que le n le plus approprié. Nous avons pu, grâce aux mesures implémentées (voir section 4.3), connaître l'influence d'un changement de la valeur du n ou des seuils de distances. Nous remarquons qu'un n d'une valeur de trois ou quatre a une influence positive sur le regroupement et que les seuils de distance utilisés par l'algorithme varient de façon assez importante selon la technique de calcul de distance entre deux vecteurs de caractéristiques qui est utilisée. Les trois variations de l'algorithme Malheur semblent résulter en un regroupement approprié, même si la version originale de l'algorithme a tendance à regrouper les maliciels en groupes plus petits, avec plusieurs groupes d'une même famille, ce qui n'est pas le cas avec les deux variations utilisant la recherche approximative. En somme, la recherche approximative est appropriée et utile pour regrouper des maliciels lorsque que les données utilisées en entrée sont textuelles, peuvent être variables et contenir des valeurs aléatoires.

D'autre part, nous avons étudié l'application de l'algorithme à des données provenant d'analyses statique et dynamique. Nous avons pu remarquer que l'aspect des données étudié (hôte, en-tête ou réseau, voir section 4.1) a une influence sur le regroupement et qu'il serait intéressant de combiner plusieurs de ces aspects pour avoir une vision plus complète d'un maliciel et mieux les regrouper. Cependant, le regroupement effectué reste limité par la qualité des données utilisées. En effet, il reste impossible de regrouper certains maliciels pour lesquels l'analyse est difficile tels que les maliciels prévus pour s'exécuter à un moment précis, ceux qui attendent une certaine durée avant de s'exécuter, ceux qui reconnaissent l'environnement d'exécution et ne s'exécutent pas s'ils se trouvent dans un bac à sable, ceux qui attendent une action de l'utilisateur, *etc.*

De nombreuses améliorations restent à faire pour raffiner le regroupement de ma-

liciels, notamment au niveau des données et des aspects choisis. En effet, la prise en compte des trois aspects lors d'un seul regroupement permettrait d'avoir des groupes de meilleure qualité, donnant une meilleure attribution d'un maliciel à un groupe.

De même, une façon d'améliorer le regroupement serait aussi de pouvoir déterminer quelles actions sont malicieuses ou non et faire en sorte que celles qui le sont aient plus de poids dans le regroupement. Cependant, une action bénigne dans un cas peut être malicieuse si combinée avec d'autres actions. Par exemple, lister les fichiers d'un système est une action tout à fait normale, tout comme encrypter un fichier, opération nécessaire pour beaucoup d'applications qui traitent des données sensibles. Un rançongiciel effectue lui aussi ces deux actions, mais à des fins malicieuses. Il est donc difficile de savoir quelles actions doivent avoir plus ou moins de poids dans le regroupement. Une façon d'aborder ce problème pourrait être de pondérer des suites d'actions au lieu de pondérer des actions seules, mais que ce soit pour des actions seules ou pour des suites d'actions, un obstacle est qu'il peut y avoir un nombre infini d'actions ou de suites d'actions, ce qui rend impossible l'énumération de l'ensemble complet.

De plus, il serait possible d'étudier un meilleur système d'extraction de signatures pour chaque groupe. Les signatures présentement extraites ne tiennent pas compte de l'ordre dans lequel les actions ont été effectuées par les maliciels du groupe. De plus, le calcul de fréquence des instructions utilisé pour extraire ces signatures se base sur l'instruction exacte et non une égalité approximative. Puisque le regroupement utilise la recherche approximative pour former les groupes, il serait intéressant d'utiliser une technique similaire pour extraire les signatures à peu près égales. De plus, de nombreuses signatures récoltées dans les expériences étaient composées d'instructions communes, effectuées par un grand nombre de familles de maliciels. Il faudrait donc se concentrer sur une extraction de signature qui permet de représenter les éléments d'un groupe assez précisément de façon à ce que tous les éléments du groupe puissent être décrits par la signature, mais que cette dernière ne puisse pas décrire des éléments d'autres groupes. Aussi, la traduction des signatures extraites en un langage reconnu par la communauté de cybersécurité comme par exemple Snort [8] ou Yara [9] aurait

8. https://www.snort.org/
9. http://yararules.com/

plusieurs avantages :

— un analyste ou un système automatisé pourrait utiliser ces signatures pour reconnaître un maliciel facilement ;

— selon le langage utilisé par les signatures, elles pourraient être directement utilisées par des systèmes de détection d'intrusion (IDS).

L'écosystème des maliciels étant un environnement en évolution continue, il serait intéressant d'étudier le regroupement évolutif des maliciels, pour que seuls les nouveaux éléments soient ajoutés aux groupes plutôt que de devoir recommencer le regroupement. Ce système évolutif devrait prendre en compte l'âge d'un maliciel pour permettre de retirer les éléments trop anciens du regroupement, évitant ainsi que les groupes deviennent trop gros ou trop nombreux.

Pour rendre l'algorithme plus rapide et plus performant, l'utilisation de calcul distribué pourrait être mise en place, par exemple en utilisant Apache Spark™ [10], cadre efficace pour des calculs complexes sur de nombreuses données.

En outre, l'utilisation d'une technologie de visualisation différente pourrait faciliter l'étude du graphe résultant du regroupement et même permettre de comparer automatiquement deux graphes, de rassembler visuellement les groupes appartenant à la même famille ou même d'attirer l'attention sur les noeuds dont la famille ne correspond pas au reste du groupe. En effet, Gephi permet d'organiser le graphe en groupes visibles, mais ce dernier ne permet pas de choisir comment ces derniers sont organisés. Cela veut dire que si le regroupement a généré deux groupes dont la plupart des éléments ont une seule et même famille, les groupes ne sont pas forcément à côté l'un de l'autre après organisation du graphe. Un rassemblement par famille permettrait une analyse plus facile par un humain. De même, il n'y a pas d'outils pour la recherche d'éléments mal étiquetés ou mal regroupés, ce qui rend difficile l'analyse de ces cas.

Les applications d'un tel algorithme de regroupement sont nombreuses, que ce soit dans le domaine des maliciels ou non. À terme, un tel système pourrait faciliter la reconnaissance des maliciels et empêcher les nouveaux maliciels au comportement connu de passer à travers les défenses existantes.

10. https://spark.apache.org/

Annexe A

Résultats détaillés ou complémentaires

A.1 Tableaux des expériences

Les résultats complets des expériences complètes sont présentés dans les tableaux A.1 et A.2.

A.2 Graphes

Les graphes des regroupements générés par les différentes variations de l'algorithme avec les seuils ajustés comme indiqué dans le tableau 4.11 sont présentés dans cette section pour chaque aspect étudié.

tableau A.1 – Expériences réalisées pour les ajustements des seuils, 1/2

Aspect	Variation	Échantillon	Taille n-gramme	S1	S2	S3	Seuil de rejet	Nombre de prototypes	Nombre de groupes	Exactitude	Silhouette
En-tête	Dissimilitude cosinus	10000	2	0.65	0.95	0.68	10	234	21	0.738	0.114
En-tête	MD5	10000	2	0.65	0.95	0.68	10	4281	127	0.614	0.977
En-tête	Original	10000	2	0.65	0.95	0.68	10	4281	127	0.616	0.978
En-tête	Dissimilitude cosinus	10000	2	0.65	0.95	0.68	4	234	27	0.163	0.114
En-tête	Dissimilitude cosinus	10000	2	0.2	0.95	0.68	10	4306	27	0.738	0.976
En-tête	Dissimilitude cosinus	10000	2	0.4	0.95	0.68	10	2409	24	0.605	0.538
En-tête	Original	10000	2	0.2	0.95	0.68	10	4385	126	0.628	0.992
En-tête	Original	10000	2	0.4	0.95	0.68	10	4340	126	0.621	0.984
En-tête	Original	10000	2	0.8	0.95	0.68	10	4217	129	0.614	0.953
En-tête	Original	10000	2	0.65	0.2	0.68	10	4281	115	0.625	0.982
En-tête	Original	10000	2	0.65	0.4	0.68	10	4281	116	0.625	0.983
En-tête	Original	10000	2	0.65	0.6	0.68	10	4280	115	0.624	0.983
En-tête	Original	10000	2	0.65	0.8	0.68	10	4281	117	0.623	0.982
En-tête	Original	10000	2	0.65	0.95	0.2	10	4281	127	0.619	0.977
En-tête	Dissimilitude cosinus	10000	2	0.8	0.95	0.68	10	70	19	0.040	0.060
En-tête	Original	10000	2	0.65	0.95	0.4	10	4281	127	0.619	0.977
En-tête	Original	10000	2	0.65	0.95	0.6	10	4281	127	0.619	0.978
En-tête	Original	10000	2	0.65	0.95	0.8	10	4281	127	0.619	0.977
En-tête	Original	10000	2	0.65	0.95	0.95	10	4281	127	0.618	0.977
En-tête	Original	10000	2	0.65	0.95	0.68	5	4281	360	0.643	0.975
En-tête	Original	10000	4	0.65	0.95	0.68	10	4325	120	0.625	0.977
En-tête	Original	10000	2	0.8	0.5	0.4	5	4217	348	0.632	0.955
En-tête	Original	10000	2	0.65	0.95	0.68	1	4281	3985	0.788	0.974
En-tête	Dissimilitude cosinus	10000	2	0.65	0.95	0.68	1	234	34	0.164	0.114
En-tête	Original	10000	2	0.98	0.95	0.68	10	3727	160	0.552	0.784
Hôte	Original	10000	2	0.65	0.95	0.68	10	1784	113	0.509	0.338
En-tête	Dissimilitude cosinus	10000	2	0.65	0.2	0.68	10	234	85	0.144	0.122
Hôte	Original	10000	4	0.65	0.95	0.68	10	2532	124	0.650	0.317
En-tête	Dissimilitude cosinus	10000	2	0.65	0.4	0.68	10	234	85	0.144	0.122
Hôte	Original	10000	3	0.65	0.95	0.68	10	2135	123	0.570	0.348
En-tête	SSDeep	10000	2	0.65	0.95	0.68	10	4259	372	0.774	0.954
Réseau	Original	500	2	0.65	0.95	0.68	1	472	423	0.992	0.518
Réseau	Dissimilitude cosinus	500	2	0.65	0.95	0.68	1	13	2	0.336	0.456
En-tête	Dissimilitude cosinus	10000	2	0.65	0.6	0.68	10	234	85	0.144	0.122
En-tête	Dissimilitude cosinus	10000	2	0.65	0.8	0.68	10	234	49	0.156	0.134
En-tête	Dissimilitude cosinus	10000	2	0.65	0.95	0.2	10	234	21	0.161	0.114
En-tête	Dissimilitude cosinus	10000	2	0.65	0.95	0.4	10	234	21	0.161	0.114
En-tête	Dissimilitude cosinus	10000	2	0.65	0.95	0.8	10	234	21	0.161	0.114
En-tête	Dissimilitude cosinus	10000	2	0.65	0.95	0.95	10	234	21	0.161	0.114
Hôte	Original	10000	2	0.2	0.95	0.68	10	3840	115	0.853	0.442
Hôte	Original	10000	2	0.4	0.95	0.68	10	2556	115	0.712	0.415
Hôte	Original	10000	2	0.8	0.95	0.68	10	1382	107	0.574	0.293
Hôte	Original	10000	2	0.65	0.2	0.68	10	1784	118	0.408	0.385
Hôte	Original	10000	2	0.65	0.4	0.68	10	1787	119	0.443	0.367
Hôte	Original	10000	2	0.65	0.6	0.68	10	1790	118	0.416	0.389
Hôte	Original	10000	2	0.65	0.8	0.68	10	1789	111	0.443	0.356
Hôte	Original	10000	2	0.65	0.95	0.2	10	1790	108	0.493	0.363
Hôte	Original	10000	2	0.65	0.95	0.4	10	1785	112	0.500	0.371
Hôte	Original	10000	2	0.65	0.95	0.8	10	1787	106	0.513	0.344
Hôte	Original	10000	2	0.65	0.95	0.95	10	1789	107	0.493	0.355
En-tête	SSDeep	10000	2	0.2	0.95	0.68	1	4402	370	0.781	0.992
Hôte	Dissimilitude cosinus	10000	2	0.65	0.95	0.68	10	487	28	0.238	-0.122
En-tête	SSDeep	10000	2	0.4	0.95	0.68	1	4353	372	0.782	0.981
Hôte	Original	10000	2	0.65	0.95	0.68	1	1790	749	0.549	0.326
Hôte	Dissimilitude cosinus	10000	2	0.2	0.95	0.68	10	4180	33	0.850	0.159
Hôte	Dissimilitude cosinus	10000	2	0.4	0.95	0.68	10	1360	32	0.514	0.155
Hôte	Dissimilitude cosinus	10000	2	0.8	0.95	0.68	10	126	21	0.106	-0.140
Hôte	Dissimilitude cosinus	10000	2	0.65	0.2	0.68	10	479	50	0.144	-0.126
Hôte	Dissimilitude cosinus	10000	2	0.65	0.4	0.68	10	476	49	0.160	-0.104
Hôte	Dissimilitude cosinus	10000	2	0.65	0.6	0.68	10	476	49	0.160	-0.104
Hôte	Dissimilitude cosinus	10000	2	0.65	0.8	0.68	10	486	43	0.205	-0.131
Hôte	Dissimilitude cosinus	10000	2	0.65	0.95	0.2	10	476	25	0.199	-0.138

tableau A.2 – Expériences réalisées pour les ajustements des seuils, 2/2

Aspect	Variation	Échantillon	Taille n-gramme	S1	S2	S3	Seuil de rejet	Nombre de prototypes	Nombre de groupes	Exactitude	Silhouette
Hôte	Dissimilitude cosinus	10000	2	0.65	0.95	0.4	10	458	22	0.185	-0.124
Hôte	Dissimilitude cosinus	10000	2	0.65	0.95	0.8	10	486	27	0.225	-0.158
Hôte	Dissimilitude cosinus	10000	2	0.65	0.95	0.95	10	496	29	0.232	-0.151
Réseau	SSDeep	500	2	0.65	0.95	0.68	1	284	255	0.814	0.554
Réseau	Dissimilitude cosinus	500	2	0.4	0.6	0.9	1	119	34	0.920	0.601
En-tête	SSDeep	10000	2	0.8	0.95	0.68	10	3150	98	0.657	0.648
Hôte	SSDeep	10000	2	0.65	0.95	0.68	1	1119	82	0.563	0.176
En-tête	SSDeep	10000	2	0.65	0.2	0.68	10	4272	115	0.584	0.972
En-tête	SSDeep	10000	2	0.65	0.4	0.68	10	4272	115	0.584	0.972
En-tête	SSDeep	10000	2	0.65	0.95	0.2	10	4259	98	0.755	0.954
En-tête	SSDeep	10000	2	0.65	0.95	0.5	10	4259	100	0.760	0.954
Hôte	Dissimilitude cosinus	10000	2	0.65	0.95	0.68	1	486	46	0.230	-0.156
En-tête	SSDeep	10000	2	0.65	0.95	0.9	10	4259	98	0.755	0.954
Hôte	Dissimilitude cosinus	10000	3	0.65	0.95	0.68	10	497	24	0.220	-0.140
Hôte	Dissimilitude cosinus	10000	2	0.4	0.6	0.9	10	1360	101	0.464	0.159
En-tête	SSDeep	10000	2	0.65	0.95	0.68	1	4272	371	0.775	0.962
Hôte	Dissimilitude cosinus	10000	2	0.2	0.6	0.9	10	4181	109	0.813	0.259
Réseau	Dissimilitude cosinus	500	2	0.2	0.95	0.68	1	445	3	0.992	0.556
Réseau	Dissimilitude cosinus	500	2	0.4	0.95	0.68	1	119	2	0.918	0.600
Réseau	Dissimilitude cosinus	500	2	0.8	0.95	0.68	1	10	2	0.455	0.459
Réseau	Dissimilitude cosinus	500	2	0.3	0.2	0.68	1	157	157	0.972	0.618
Réseau	Dissimilitude cosinus	500	2	0.3	0.4	0.68	1	157	125	0.972	0.618
Réseau	Dissimilitude cosinus	500	2	0.3	0.6	0.68	1	157	35	0.972	0.618
Réseau	Dissimilitude cosinus	500	2	0.3	0.8	0.68	1	157	10	0.972	0.618
Réseau	Dissimilitude cosinus	500	2	0.3	0.5	0.2	1	157	110	0.972	0.618
Réseau	Dissimilitude cosinus	500	2	0.3	0.5	0.4	1	157	110	0.972	0.618
Réseau	Dissimilitude cosinus	500	2	0.3	0.5	0.8	1	157	110	0.972	0.618
Réseau	Dissimilitude cosinus	500	2	0.3	0.45	0.68	1	157	116	0.972	0.618
Réseau	SSDeep	500	2	0.2	0.95	0.68	1	452	256	0.994	0.675
Réseau	SSDeep	500	2	0.4	0.95	0.68	1	186	256	0.964	0.397
Réseau	SSDeep	500	2	0.8	0.95	0.68	1	274	255	0.801	0.530
Réseau	SSDeep	500	2	0.65	0.2	0.68	1	285	285	0.814	0.557
Réseau	SSDeep	500	2	0.65	0.4	0.68	1	285	285	0.816	0.552
Réseau	SSDeep	500	2	0.65	0.6	0.68	1	285	285	0.814	0.557
Réseau	SSDeep	500	2	0.65	0.8	0.68	1	285	275	0.816	0.552
Réseau	SSDeep	500	2	0.65	0.95	0.2	1	285	255	0.815	0.557
Réseau	SSDeep	500	2	0.65	0.95	0.4	1	285	255	0.814	0.555
Réseau	SSDeep	500	2	0.65	0.95	0.8	1	287	254	0.814	0.463
Réseau	SSDeep	500	2	0.3	0.8	0.8	1	431	278	0.992	0.542
Réseau	SSDeep	500	2	0.1	0.8	0.8	1	466	278	0.996	0.825
Réseau	SSDeep	500	4	0.1	0.8	0.8	1	490	311	0.998	0.711
Réseau	SSDeep	500	4	0.65	0.8	0.8	1	318	301	0.840	0.497
En-tête	Dissimilitude cosinus	10000	2	0.3	0.5	0.8	10	4073	178	0.746	0.901
Hôte	SSDeep	10000	2	0.1	0.8	0.8	10	8443	471	0.988	0.417
Réseau	Original	500	2	0.2	0.95	0.68	1	491	423	0.996	0.949
Réseau	Original	500	2	0.4	0.95	0.68	1	487	422	0.992	0.837
Réseau	Original	500	2	0.8	0.95	0.68	1	445	417	0.980	0.305
Réseau	Original	500	2	0.65	0.2	0.68	1	472	472	0.992	0.518
Réseau	Original	500	2	0.65	0.4	0.68	1	472	472	0.992	0.518
Réseau	Original	500	2	0.65	0.6	0.68	1	474	474	0.992	0.540
Réseau	Original	500	2	0.65	0.8	0.68	1	474	449	0.992	0.540
Réseau	Original	500	2	0.65	0.95	0.2	1	472	423	0.992	0.518
Réseau	Original	500	2	0.65	0.95	0.4	1	472	423	0.992	0.518
Réseau	Original	500	2	0.65	0.95	0.8	1	472	423	0.992	0.518
Réseau	Original	500	4	0.65	0.95	0.68	1	487	483	0.992	0.819
Réseau	Dissimilitude cosinus	500	4	0.3	0.5	0.8	1	101	35	0.902	0.718
Réseau	Dissimilitude cosinus	500	3	0.3	0.5	0.8	1	119	45	0.918	0.666
En-tête	SSDeep	10000	4	0.65	0.4	0.68	10	2432	155	0.400	0.503
Hôte	SSDeep	10000	2	0.2	0.95	0.68	10	2257	65	0.950	0.243
Hôte	SSDeep	10000	2	0.4	0.95	0.68	10	3596	89	0.837	0.221
Hôte	SSDeep	10000	2	0.8	0.95	0.68	10	277	46	0.350	0.128
Hôte	SSDeep	10000	2	0.65	0.2	0.68	10	1152	154	0.601	0.208
Hôte	SSDeep	10000	2	0.65	0.4	0.68	10	1142	152	0.532	0.178
Hôte	SSDeep	10000	2	0.65	0.6	0.68	10	1155	152	0.615	0.200
Hôte	SSDeep	10000	2	0.65	0.8	0.68	10	1070	106	0.526	0.166
Hôte	SSDeep	10000	2	0.65	0.95	0.2	10	1110	60	0.588	0.162
Hôte	SSDeep	10000	2	0.65	0.95	0.4	10	1110	60	0.588	0.162
Hôte	SSDeep	10000	2	0.65	0.95	0.8	10	1142	70	0.685	0.194

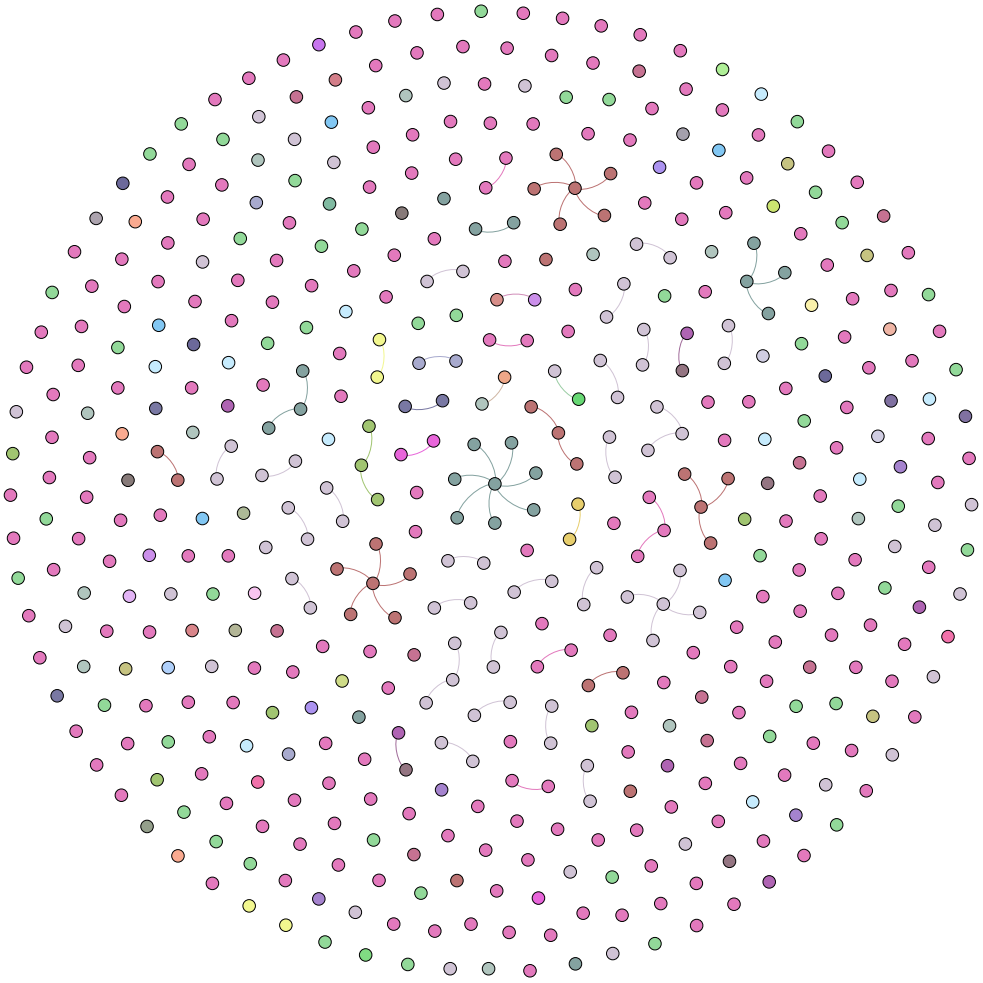


figure A.1 – Graphe généré avec l'algorithme original avec les seuils ajustés pour l'aspect réseau (échantillon = 500)

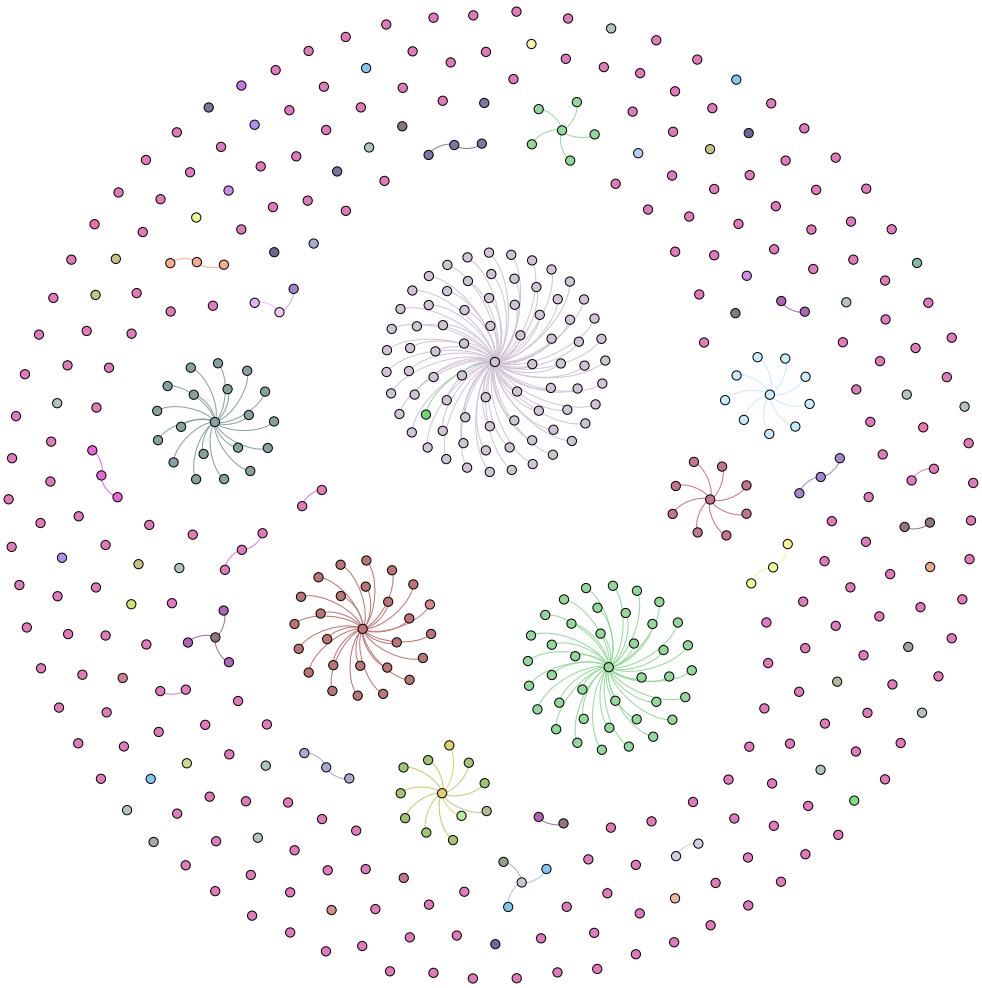


figure A.2 – Graphe généré avec l'algorithme utilisant SSDeep avec les seuils ajustés pour l'aspect réseau (échantillon = 500)

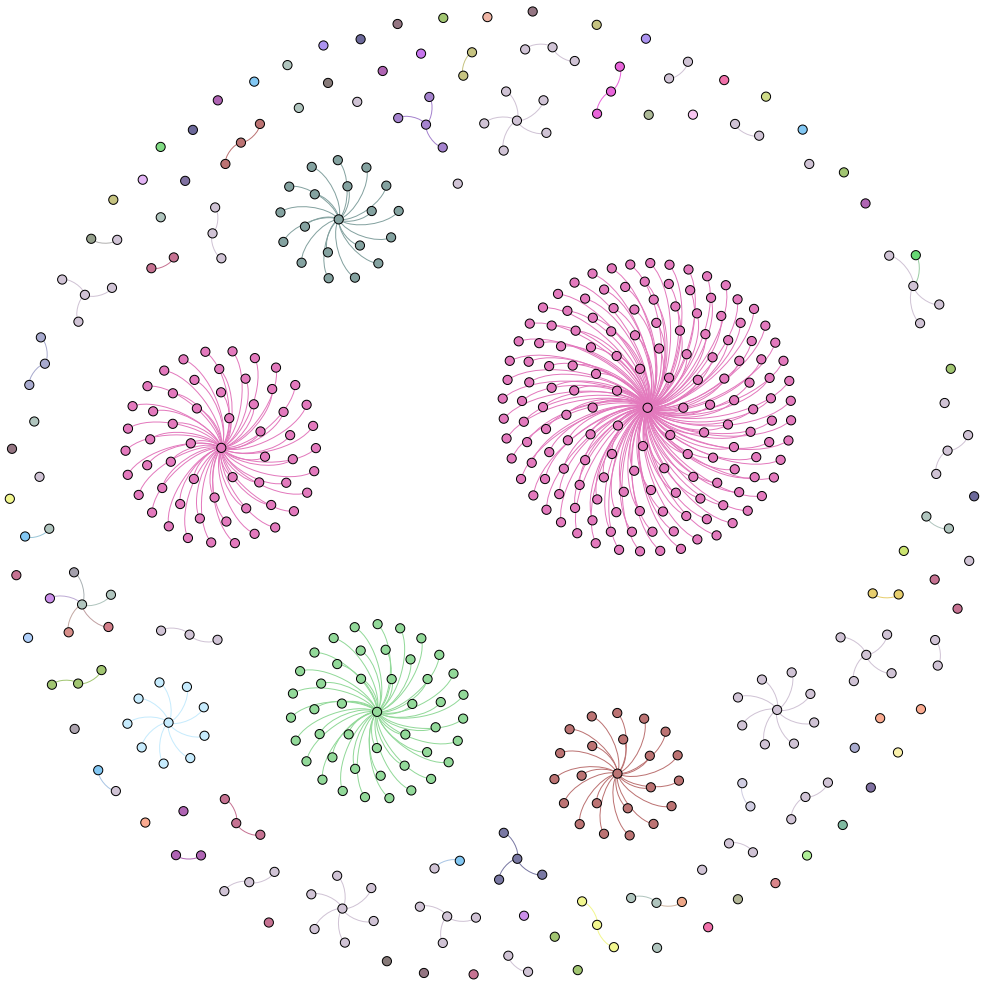


figure A.3 – Graphe généré avec l'algorithme utilisant la dissimilitude cosinus avec les seuils ajustés pour l'aspect réseau (échantillon = 500)

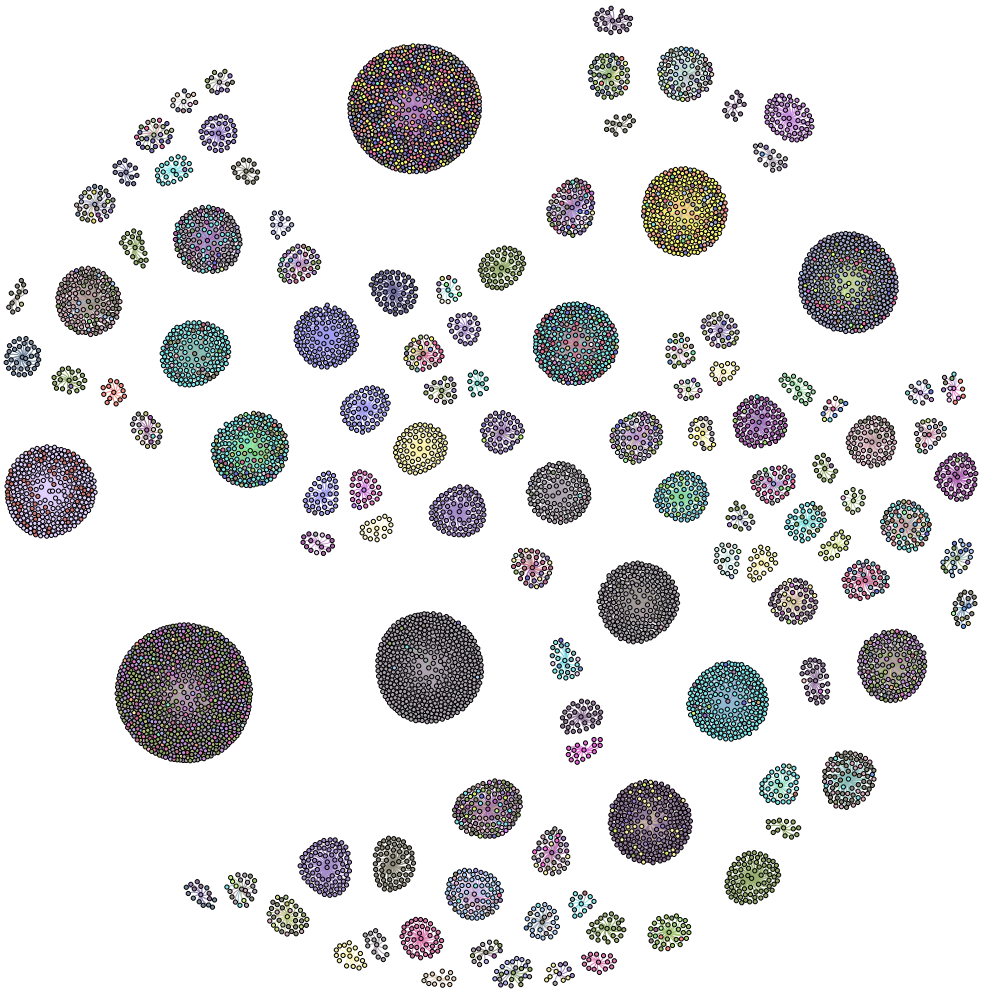


figure A.4 – Graphe généré avec l'algorithme original avec les seuils ajustés pour l'aspect hôte (échantillon = 10 000)

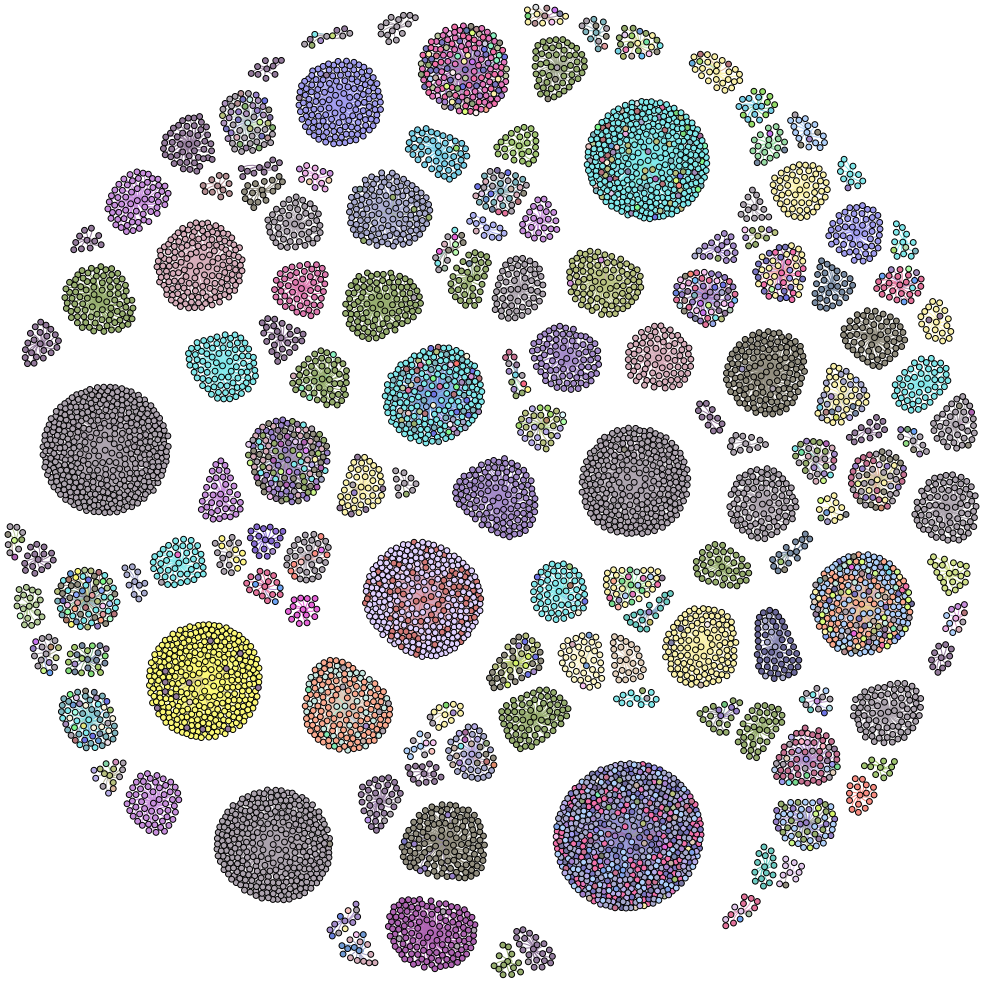


figure A.5 – Graphe généré avec l'algorithme utilisant SSDeep avec les seuils ajustés pour l'aspect hôte (échantillon = 10 000)

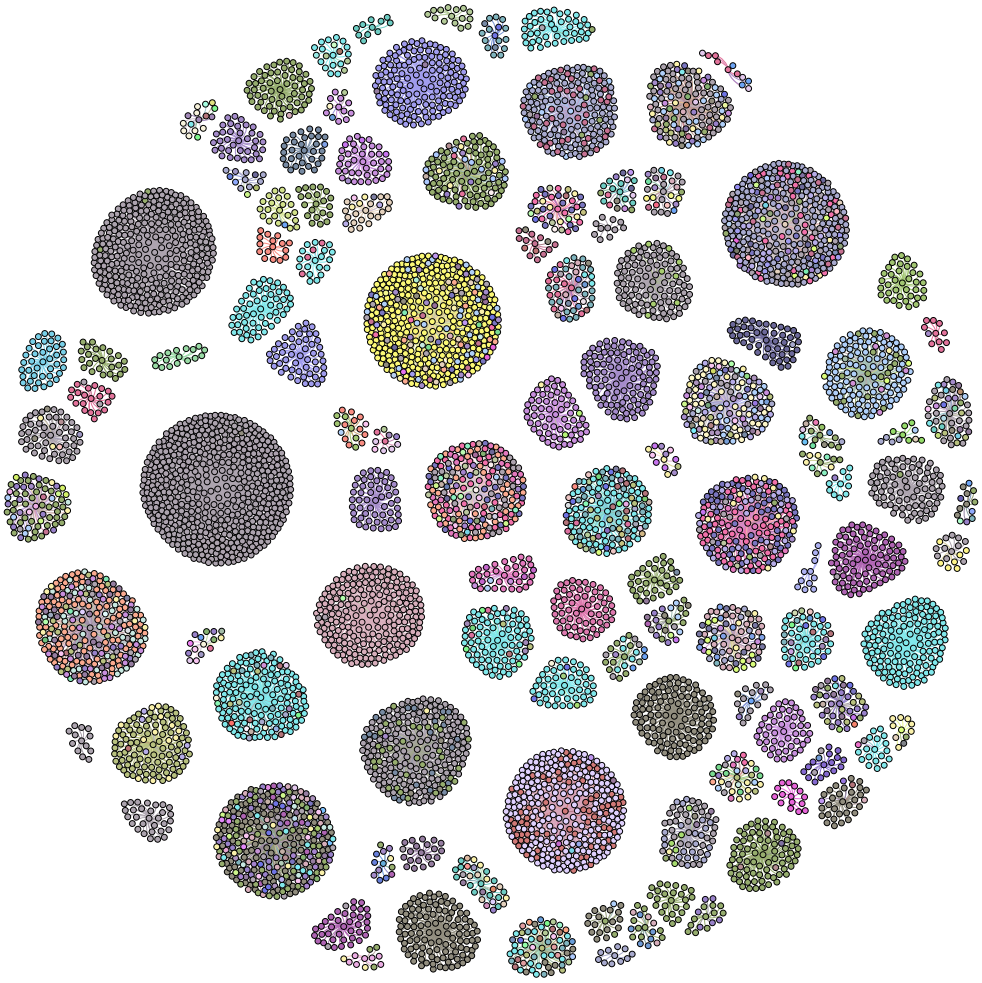


figure A.6 – Graphe généré avec l'algorithme utilisant la dissimilitude cosinus avec les seuils ajustés pour l'aspect hôte (échantillon = 10 000)

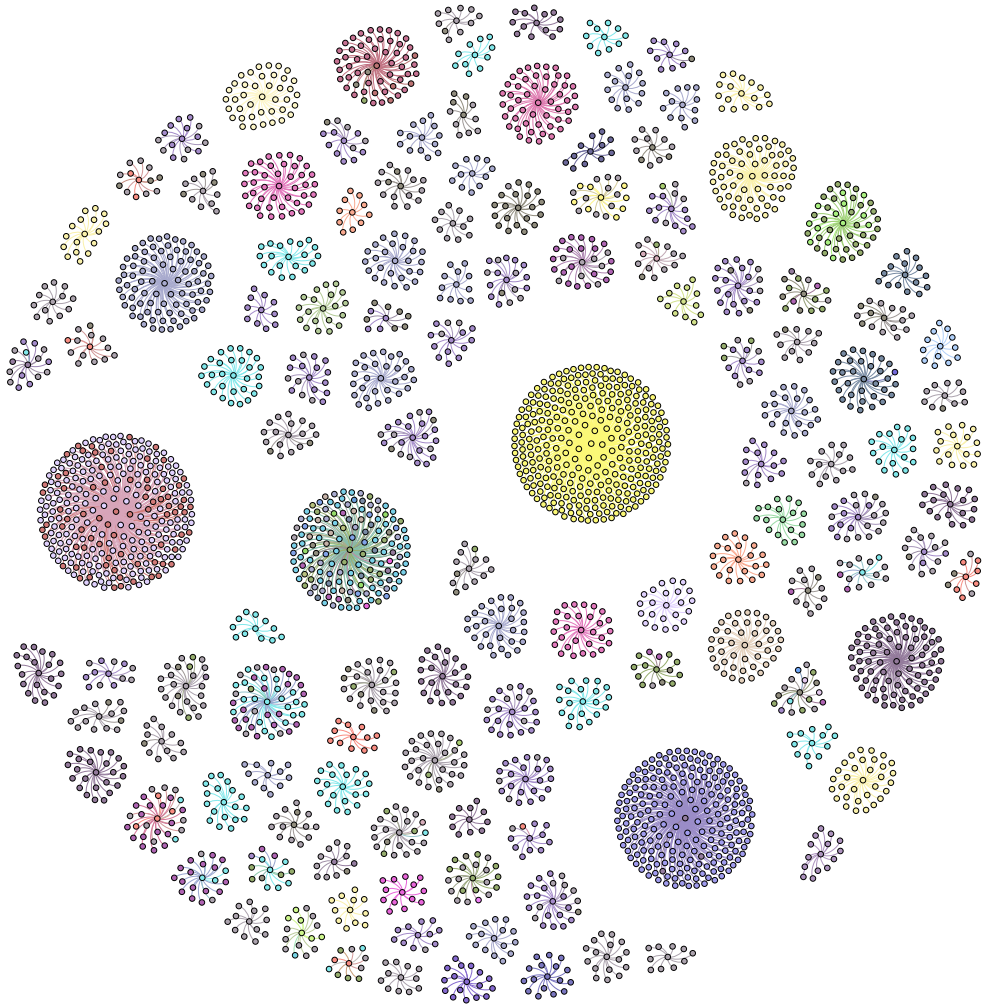


figure A.7 – Graphe généré avec l'algorithme original avec les seuils ajustés pour l'aspect en-tête (échantillon = 10 000)

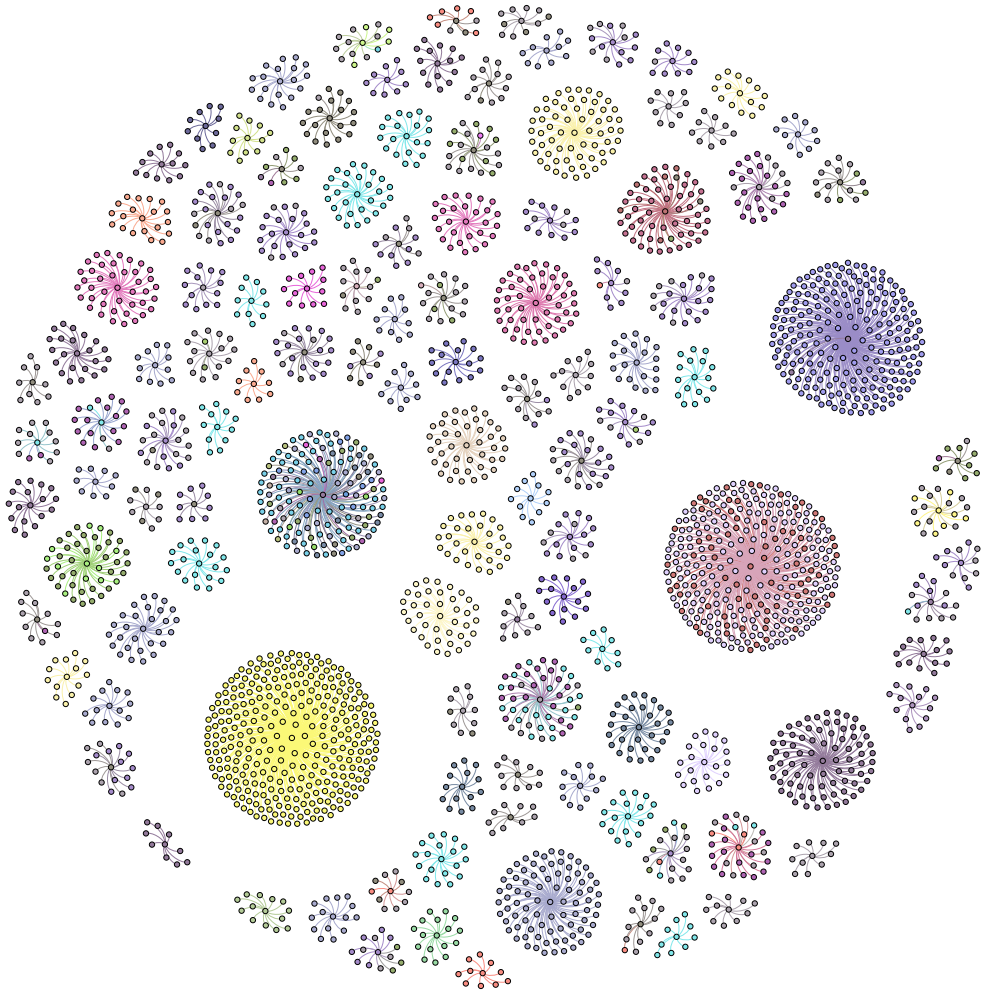


figure A.8 – Graphe généré avec l'algorithme utilisant SSDeep avec les seuils ajustés pour l'aspect en-tête (échantillon = 10 000)

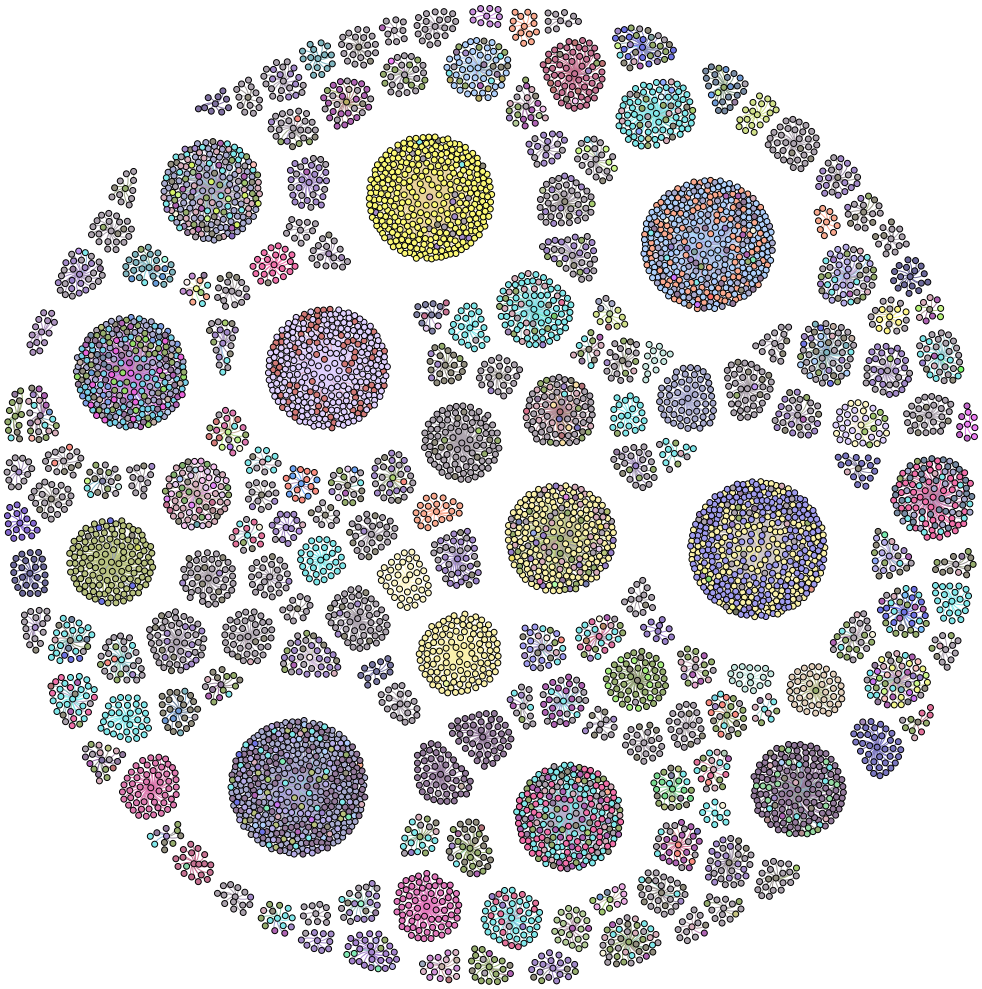


figure A.9 – Graphe généré avec l'algorithme utilisant la dissimilitude cosinus avec les seuils ajustés pour l'aspect en-tête (échantillon = 10 000)

Bibliographie

[1] Charlie Anthe et 30 autres auteurs : Microsoft security intelligence report, volume 21, 2016. 180 pages.

[2] Michael Bailey, Jon Oberheide, Jon Andersen, Z. Morley Mao, Farnam Jahanian et Jose Nazario : Automated classification and analysis of internet malware. Dans Christopher Kruegel, Richard Lippmann et Andrew Clark, éditeurs : Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, volume 4637, pages 178–197. Springer, Berlin, Heidelberg, 2007.

[3] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel et Engin Kirda : Scalable, behavior-based malware clustering. Dans Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, 2009. 18 pages.

[4] Deepayan Chakrabarti, Ravi Kumar et Andrew Tomkins : Evolutionary clustering. Dans Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 554–560, Philadelphia, PA, 2006.

[5] Moses Charikar, Chandra Chekuri, Tomás Feder et Rajeev Motwani : Incremental clustering and dynamic information retrieval. SIAM Journal on Computing, 33(6):1417–1440, 2004.

[6] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino et Belle L Tseng : Evolutionary spectral clustering by incorporating temporal smoothness. Dans Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 153–162, San Jose, CA, 2007.

[7] Ekta Gandotra, Divya Bansal et Sanjeev Sofat : Malware analysis and classification : A survey. Journal of Information Security, 5(2):56–64, 2014.

[8] Weiwei Hu et Ying Tan : Partitioning based n-gram feature selection for malware classification. Dans Ying Tan et Yuhui Shi, éditeurs : Data Mining and Big Data, Lecture Notes in Computer Science, volume 9714, pages 187–195. Springer, Berlin, Heidelberg, 2016.

[9] Rafiqul Islam, Ronghua Tian, Lynn M Batten et Steve Versteeg : Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications, 36(2):646–656, 2013.

[10] Anil K Jain, M Narasimha Murty et Patrick J Flynn : Data clustering : A review. ACM Computing Surveys, 31(3):264–323, 1999.

[11] Saeed Nari et Ali A Ghorbani : Automated malware classification based on network behavior. Dans Proceedings of the 2013 International Conference on Computing, Networking and Communications, pages 642–647, San Diego, CA, 2013.

[12] Mathias Payer : Embracing the new threat : Towards automatically self-diversifying malware. Dans Symposium on Security for Asia Network, Singapore, Singapore, 2014. 5 pages.

[13] Roberto Perdisci, Davide Ariu et Giorgio Giacinto : Scalable fine-grained behavioral clustering of HTTP-based malware. Computer Networks, 57(2):487–500, 2013.

[14] Roberto Perdisci, Wenke Lee et Nick Feamster : Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. Dans Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, pages 391–404, San Jose, CA, 2010.

[15] Slobodan Petrovic : A comparison between the Silhouette index and the Davies-Bouldin index in labelling IDS clusters. Dans Proceedings of the 11th Nordic Workshop of Secure IT Systems, pages 53–64, Linkping, Sweden, 2006.

[16] M Zubair Rafique et Juan Caballero : FIRMA : Malware clustering and network signature generation with mixed network behaviors. Dans Salvatore J. Stolfo, Angelos Stavrou et Charles V. Wright, éditeurs : Research in Attacks, Intrusions, and Defenses, Lecture Notes in Computer Science, volume 8145, pages 144–163. Springer, Berlin, Heidelberg, 2013.

[17] Konrad Rieck, Philipp Trinius, Carsten Willems et Thorsten Holz : Automatic analysis of malware behavior using machine learning. Journal of Computer Security, 19(4):639–668, 2011.

[18] Matthew G Schultz, Eleazar Eskin, Erez Zadok et Salvatore J Stolfo : Data mining methods for detection of new malicious executables. Dans Proceedings of the 2001 IEEE Symposium on Security and Privacy, pages 38–49, Oakland, CA, 2001.

[19] Marcos Sebastián, Richard Rivera, Platon Kotzias et Juan Caballero : Avclass : A tool for massive malware labeling. Dans Fabian Monrose, Marc Dacier, Gregory Blanc et Joaquin Garcia-Alfaro, éditeurs : Research in Attacks, Intrusions, and Defenses, Lecture Notes in Computer Science, volume 9854, pages 230–253. Springer, Berlin, Heidelberg, 2016.

[20] Andrew Walenstein et Arun Lakhotia : The software similarity problem in malware analysis. Dans Rainer Koschke, Ettore Merlo et Andrew Walenstein, éditeurs : Duplication, Redundancy, and Similarity in Software, Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. 10 pages.

[21] Andrew Walenstein, Michael Venable, Matthew Hayes, Christopher Thompson et Arun Lakhotia : Exploiting similarity between variants to defeat malware : “Vilo” method for comparing and searching binary programs. White Paper for BlackHat DC, 2007. 12 pages.

[22] Wei Yan, Zheng Zhang et Nirwan Ansari : Revealing packed malware. IEEE Security & Privacy, 6(5):65–69, 2008.

[23] Qinghua Zhang et Douglas S Reeves : MetaAware : Identifying metamorphic

malware. Dans Proceedings of the 23rd Annual Computer Security Applications Conference, pages 411–420, Miami Beach, FL, 2008.